Краткое содержание

Оглавление

1 Как мы сюда попали?

Приступим

Недавно я прочел высказывание одного разработчика для платформы Mozilla, где говорится о той напряженности, которая всегда сопутствует разработке стандартов (http://lists.w3.org/Archives/Public/public-html/2010Jan/0107.html):

> Спецификация и программная реализация должны пройти путь осторожного совместного развития. Ведь, с одной стороны, не хочется, чтобы реализация появилась раньше, чем выйдет окончательная версия стандарта, иначе разработчики станут принимать во внимание особенности конкретной программы, то есть в дело пойдет лишь фрагмент спецификации. С другой стороны, не хочется утверждать полный стандарт до выхода хотя бы одной реализации, иначе придется обойтись без обратной связи, в частности, не будут приняты в расчет мнения создателей программ. Противоречие неизбежно. Нам остается решать проблему методом проб и ошибок.

Пока я буду рассказывать, как появился HTML5, помните эти слова.

MIME-типы

Книга, которую вы держите в руках, посвящена HTML5, а не предшествующим версиям стандарта HTML и тем более не XHTML. Но, чтобы ясно проследить историю HTML5 и увидеть причины, предшествующие его появлению, нужно сначала овладеть кое-какими техническими деталями, в частности получить понятие о MIME-типах.

Каждый раз, когда ваш браузер пытается загрузить страницу, сервер, прежде чем отослать клиентской программе код самой страницы, отправляет ей ряд заголовков. Пользователь обычно не видит этих заголовков, хотя в некоторых программах для веб-разработчиков предусматривается возможность их отображения. Заголовки важны постольку, поскольку они сообщают браузеру, как воспринимать код посылаемой вслед за ними страницы. Самый информативный заголовок называется `Content-Type` и выглядит, например, так:

```
Content-Type: text/html
```

Значение `text/html` называется типом содержимого, или MIME-типом загружаемой страницы. Только данный заголовок определяет, каково содержание

отдельного ресурса и, следовательно, как этот ресурс должен выводиться на экран. У изображений собственные MIME-типы (`image/jpeg` — для картинок в формате JPEG, `image/png` — для формата PNG и т. д.). Собственными MIME-типами оснащены файлы JavaScript, таблицы стилей CSS и, в общем-то, все, что есть в Сети.

На самом деле все чуть сложнее, чем рассказано выше. Самые ранние веб-серверы, под которыми я понимаю веб-серверы 1993 года и нескольких последующих лет, не отправляли заголовки `Content-Type`, потому что те были изобретены только в 1994 году. Ради совместимости, во имя которой, кстати, с 1993 года по сей день делалось и делается очень много всего, отдельные популярные браузеры при определенных условиях игнорируют заголовки `Content-Type`. Это называется контент-сниффингом. Но общее правило таково, что любой фрагмент содержимого Сети, будь то HTML-страница, изображение, сценарий, видеозапись, PDF-документ или что-то еще под собственным URL-адресом, посылается клиентской программе с предварительным уведомлением о MIME-типе в заголовке `Content-Type`.

Хорошенько запомните эту информацию, так как она еще пригодится.

Большое отступление о том, как появляются стандарты

Откуда взялся тег `<img>`? Не думаю, что вы хоть иногда задавались подобным вопросом. Очевидно, *кто-то* его создал. Такие вещи не берутся ниоткуда. Из всех элементов и атрибутов HTML, которыми вы в разное время пользовались, абсолютно каждый был когда-то кем-то создан. Этот кто-то придумал, как должен работать элемент или атрибут, и письменно сформулировал свои мысли. Такого рода люди, бесспорно, умнее нас с вами, но они тоже обычные люди.

Если стандарт разрабатывался открыто, то можно вернуться в прошлое и увидеть, как рождалась идея того или иного пункта спецификации. Обсуждения ведутся в почтовых рассылках, а их архивы обычно имеют интерфейс поиска. Чтобы ответить на вопрос о теге `<img>`, я решил немного позаниматься «электронной археологией» и погрузился в толщу времен, когда еще не существовало Консорциума Всемирной паутины (W3C), а все веб-серверы мира можно было пересчитать по пальцам. Речь идет о первых днях Интернета.

25 февраля 1993 года Марк Андрессен (Marc Andreessen) написал[1]:

> Предлагаю новый опциональный HTML-тег:
> IMG
> При нем должен обязательно указываться аргумент SRC="url".
> Тег отсылает к файлу растрового изображения (bitmap или pixmap). Браузер будет запрашивать этот файл в Сети, распознавать как изображение и вставлять в текст сообразно месту тега в коде страницы.

---

[1] http://1997.webhistory.org/www.lists/www-talk.1993q1/0182.html. За ходом обсуждения, цитируемого здесь и далее, можно следить, переходя по ссылкам Next message и Previous message, расположенным вверху и внизу указанной веб-страницы. — *Примеч. авт.*

Пример использования:

```
<IMG SRC="file://foobar.com/foo/bar/blargh.xbm">
```

(Закрывающий тег не требуется.)
Как и любое другое содержимое, этот тег может быть вложет внутрь якоря. Тогда изображение станет чувствительным к активизации, как и обычная текстовая ссылка.
Следует предоставить браузерам свободу выбора графических форматов, которые будут в них поддерживаться. Удачным выбором мне представляются, например, Xbm и Xpm. Если браузер не умеет отображать данный формат, пусть он делает то, что разработчикам заблагорассудится предусмотреть на этот случай (так, в X Mosaic будет выводиться растровая картинка, замещающая нужное изображение).
Данная функциональность будет реализована в X Mosaic. Мы работаем над ней и собираемся использовать по крайней мере внутри команды разработчиков. Разумеется, я буду рад вашим предложениям по поводу того, каким должен быть механизм поддержки изображений в HTML. Если у вас появится мысль удачнее моей, поделитесь, пожалуйста. Я знаю, что разнообразие графических форматов делает ситуацию чрезвычайно туманной, но альтернативы не вижу. Можно разве что сказать: «Пусть браузер работает как умеет» — и ждать той поры, когда будет предложено идеальное решение (может быть, когда-нибудь, с помощью MIME-типов).

Эту цитату надо пояснить. Xbm и Xpm — популярные графические форматы в UNIX-системах; Mosaic — один из первых браузеров. Его версия, которая работала в UNIX-системах, называлась X Mosaic. Когда Марк отправлял это письмо на дискуссионный лист в начале 1993 года, он еще не основал компанию Mosaic Communications Corporation, которая впоследствии принесла ему известность, и еще не начал работу над флагманским продуктом будущей компании — браузером Mosaic Netscape (фирма и программа позже были переименованы в Netscape Corporation и Netscape Navigator соответственно).

Говоря о MIME-типах «может быть, когда-нибудь», Марк ссылается на предусмотренный в протоколе HTTP механизм «переговоров о содержимом». Благодаря этому механизму клиентская программа-браузер сообщает серверу (в данном случае веб-серверу), какие типы ресурсов она умеет обрабатывать (например, `image/jpeg`), а сервер в ответ может прислать содержимое в удобном для клиента формате. По состоянию на февраль 1993 года программно реализован только самый первый вариант протокола HTTP (1991 год), в котором клиент не мог передать серверу информацию о поддерживаемых типах изображений. Отсюда проблема, с которой столкнулся Марк.

Несколько часов спустя Тони Джонсон (Tony Johnson) ответил:

У меня в Midas 2.0 (программа пока находится во внутреннем пользовании SLAC, но уже готова к открытому релизу) применяется похожее решение. Тег иначе назван, и в нем есть еще один аргумент NAME="name", но функциональность абсолютно та же, что и у предложенного вами тега IMG. Пример:

```
<ICON name="NoEntry" href="http://note/foo/bar/NoEntry.xbm">
```

Смысл параметра NAME в том, чтобы позволить браузеру прибегать к помощи набора «встроенных» картинок. Если имя соответствует изображению, которым браузер уже располагает, то вместо того, чтобы доставать картинку из Сети, программа использует готовый графический файл. Кроме того, имя изображения может подсказывать текстовым браузерам, каким символом заместить картинку.

Меня мало волнуют имена тегов и параметров (но если бы мы решили прийти к компромиссу, они приобрели бы большое значение). Столь же маловажен, по-моему, вопрос об аббревиатурах, то есть почему IMG и SRC, а не IMAGE и SOURCE. Мне самому больше по душе ICON — это слово дает понять, что картинка должна быть маленькой, вроде значка. Готов признать, впрочем, что слово ICON итак уже обременено множеством смыслов.

Midas — это еще один ранний браузер, современник X Mosaic. Он был кросс-платформенным и работал как в UNIX, так и в VMS. Аббревиатура SLAC расшифровывается как Stanford Linear Accelerator Center (Научно-исследовательский центр при Стэнфордском линейном ускорителе (электронов)). Теперь этот центр получил статус национальной лаборатории. Инженеры SLAC запустили первый веб-сервер в США, который фактически был и первым за пределами Европы. В феврале 1993 года SLAC считался долгожителем Сети (работал год и три месяца!) с пятью веб-страницами на сервере.

Вот продолжение письма Тони:

Раз уж мы заговорили о новых тегах, то расскажу о другом аналогичном теге, поддержку которого я намерен реализовать в Midas 2.0. Его схема такова:

```
<INCLUDE HREF="...">
```

В код документа в месте вхождения этого тега должен быть вставлен другой документ, на который ссылается тег. Этот документ может быть чем угодно по типу содержания, но основная задача тега — обеспечить вставку изображений произвольного размера в веб-страницы. Когда будет реализован HTTP2, клиент и сервер смогут дополнительно оговаривать формат вставляемого документа.

Под названием *HTTP2* здесь фигурирует базовый HTTP в редакции 1992 года. В начале 1993 года значительная его часть не имела программной реализации. Черновой вариант, известный как HTTP2, после некоторой доработки был стандартизован в качестве HTTP 1.0. В стандарт HTTP 1.0 уже включены заголовки-запросы для переговоров о содержимом, то есть то самое «может быть, когда-нибудь» наступило довольно скоро.

Тони так заканчивает свое письмо:

Я рассматривал и следующую альтернативу:

```
<A HREF="..." INCLUDE>См. фотографию</A>
```

Не хочется прибавлять новую функциональность тегу <A>. Но такое решение было бы удобно для совместимости с браузерами, которые не понимают параметр INCLUDE. Иными словами, если браузер распознает команду INCLUDE, то он заменит текст ссылки («См. фотографию» в данном случае) картинкой, а более старый или более глупый браузер просто проигнорирует INCLUDE.

Это предложение не было реализовано, хотя идея заместительного текста на случай, если изображение отсутствует, очень привлекательна и не упоминается в предложенной Марком конструкции тега `<IMG>`. Много лет спустя идея была осуществлена в атрибуте `<img alt="...">` (после чего все испортил Netscape, который ошибочно отображал текст-заместитель в виде всплывающей подсказки).

Через несколько часов после сообщения Тони ему и Марку ответил Тим Бернерс-Ли (Tim Berners-Lee):

> Я полагал, что картинки можно представлять в виде Картинка. Значение ссылочных отношений таково:
> EMBED — встроить содержимое в данное место документа для отображения;
> PRESENT — отображать содержимое, если исходный документ доступен.
> Стоит отметить, что возможны разные сочетания атрибутов. Если браузер не поддерживает какой-то один из них, сбоя не будет. Понятно, что для создания таким способом значков, чувствительных к пользовательскому выбору, нужно вложить один якорь в другой. Но, честно говоря, я не хотел бы вводить особый тег.

Это предложение не было реализовано, но атрибут `rel` существует до сих пор (см. раздел «Элемент HEAD» главы 3).

Джим Дэвис (Jim Davis) прибавил:

> Хорошо бы еще иметь возможность указывать тип содержимого, например, так:
>
> ```
> <IMG HREF="http://nsa.gov/pub/sounds/gorby.au" CONTENT-TYPE=audio/basic>
> ```
>
> Однако я, конечно, надеюсь дожить до того времени, когда тип содержимого будет строго определяться по расширению файла.

Это предложение тоже не было реализовано, хотя позднее Netscape стал поддерживать встраивание произвольных мультимедийных объектов с помощью тега `<embed>`.

Джей Вебер (Jay C. Weber) написал следующее:

> Отображение графики в браузерах — моя давняя мечта. Но неужели для каждого вида мультимедийной информации надо создавать персональный тег? Еще недавно все с радостью ожидали появления механизма MIME-типов. Что же случилось теперь?

Марк Андрессен ответил:

> Это не альтернатива предстоящему использованию MIME-типов как стандартного механизма обработки документов. Это простая реализация функциональности, которая нужна независимо от MIME.

Джей Вебер возразил:

> Забудем на время о MIME-типах, они отвлекают от сути. Я, собственно, не согласен с вашим подходом к поддержке встроенных изображений, ведь можно ожидать, что на следующей неделе кто-нибудь предложит новый тег <AUD SRC="file://foobar.com/foo/bar/blargh.snd"> для звуковых файлов. Между тем за использование единого для всех медийных типов способа встраивания пришлось бы платить не такой уж и дорогой монетой.

Опыт свидетельствует, что беспокойство Джея было вполне обоснованным. Прошло, правда, больше недели, но в HTML5 появились теги `<video>` и `<audio>`.

В ответ на первое письмо Джея Дейв Рэггет (Dave Raggett) написал:

> Совершенно правильно! Собираюсь рассмотреть множество графических и псевдографических типов в связи с механизмом переговоров о формате. Замечание Тима о поддержке активных областей на картинках тоже учту.

В том же 1993 году, но немного позже Дейв опубликовал стандарт HTML+, задуманный им в качестве замены первоначальному HTML. Стандарт не был реализован; на смену HTML пришел ретроспективный HTML 2.0 — формальное описание того аппарата тегов, который на момент принятия стандарта уже широко использовался: «Эта спецификация сводит воедино, уточняет и формально описывает функции из того набора, который приблизительно соответствует возможностям общеупотребительного HTML по состоянию на июнь 1994 года».

Позже на основе спецификации HTML+ Дейв Рэггет создал стандарт HTML 3.0. Нигде, кроме внутренней (используемой W3C в качестве эталона) программы Arena, стандарт HTML 3.0 не был реализован. На смену ему пришел HTML 3.2 — вновь ретроспектива: «Сохраняя полную обратную совместимость с существующим HTML 2.0, стандарт HTML 3.2 добавляет к нему широко распространенные новые функции: таблицы, приложения и обтекание изображений текстом».

Еще позже Дейв выступил в качестве одного из соавторов HTML 4.0, разработал HTML Tidy, принимал участие в подготовке XHTML, XForms, MathML и других современных спецификаций W3C.

В далеком 1993 году Марк ответил Дейву так:

> Может быть, стоило бы действительно задуматься о графическом процедурном языке общего назначения, возможности которого позволили бы присоединять произвольные гиперссылки к значкам, картинкам, тексту и т. д.? Не известно ли кому-нибудь из подписчиков, как с этим обстоит дело в проекте Intermedia?

Intermedia — это гипертекстовый проект Брауновского университета. Работа над ним велась в 1985–1991 годах. Рабочей средой для Intermedia была операционная система A/UX — UNIX-подобная среда, функционирующая на компьютерах Macintosh первых поколений. Мысль о «графическом процедурном языке общего назначения» впоследствии прижилась. Современные браузеры поддерживают как SVG-графику (декларативную разметку со встроенной возможностью разработки сценариев), так и `<canvas>` (процедурный интерфейс непосредственного программирования графики). Правда, исторически `<canvas>` был проприетарным расширением для браузера и рабочая группа WHAT внесла его в спецификацию постфактум.

Билл Дженсен (Bill Janssen) сообщил:

> Функциональность, о которой вы говорите, действительно очень ценна. Кроме Intermedia, есть другие системы, в которых она реализована: Andrew и Slate. В основе Andrew лежит система вставок. Каждой вставке присвоен определенный тип: текст, растровое изображение, векторное изображение, анимация, электронное письмо, таблица и др. Реализовано рекурсивное вложение произвольной глубины, то есть вставка любого типа может быть вложена во вставку любого из типов, поддерживающих вложение. Так, например, можно поместить вставку в тексте после любого символа (если мы работаем с текстовым окном), в любой прямоугольной области (если мы работаем с графикой), в любой ячейке (если мы имеем дело с таблицей).

Andrew — сокращенное название системы пользовательского интерфейса Andrew. Ее в те годы все называли Andrew Project.

Тем временем Томас Файн (Thomas Fine) выдвинул альтернативное предложение:

> Я думаю так. Работу с изображениями в Сети лучше всего построить на системе MIME-типов. А формат Postscript, для которого наверняка уже существует особый тип, как раз позволяет совмещать текстовую и графическую информацию с большим удобством.
> «Но в нем ведь не будут работать гипертекстовые ссылки», — скажете вы. Да, это так. Однако мне кажется, что проблему решает технология Display Postscript. Если и нет, то дополнить до этого стандартный Postscript — легкая задача. Определим команду-якорь, которая бы содержала URL и интерпретировала текущий контур как замкнутую область-кнопку. Поскольку в Postscript предусмотрена прорисовка контуров, это позволяет легко делать кнопки произвольной формы.

Display PostScript — технология экранной прорисовки, совместно разработанная Adobe и NeXT.

Это предложение не было реализовано. До сих пор, впрочем, время от времени озвучивается мысль о том, что для улучшения языка HTML надо его просто чем-нибудь заменить.

2 марта 1993 года Тим Бернерс-Ли оставил такой комментарий:

> В HTTP2 документам разрешено нести любой MIME-тип, понимаемый клиентской программой, а не только какой-либо один из зарегистрированных. Это оставляет пространство для экспериментов. Думаю, Postscript с поддержкой гипертекста мог бы стать предметом таких экспериментов. Не знаю, достаточно ли функциональности у Display Postscript, но мне известно, что компания Adobe сейчас активно продвигает свой формат PDF на основе Postscript. В документах этого формата будут работать гиперссылки, но просматривать такие документы можно будет только в проприетарных программах Adobe.
> Я полагал, что обобщенный язык якорей (на основе HyTime?) позволит гипертекстовым и мультимедийным (графика/видео) стандартам развиваться независимо, что пойдет на пользу и тем и другим.
> Пусть лучше будет тег не IMG, а INCLUDE, который бы ссылался на документы произвольного типа. Или EMBED, если INCLUDE звучит как директива C++ и будут ошибочно думать, что нужен исходный SGML-код, который браузер будет разбирать (мы имеем в виду не это).

HyTime — это одна из ранних гипертекстовых систем документов, основанная на разметке SGML. В обсуждениях стандартов HTML и затем XML в 1990-е годы о ней часто вспоминали.

Предложенный Тимом тег `<INCLUDE>` так никогда и не появился, хотя отголоски этой идеи можно наблюдать в тегах `<object>`, `<embed>` и `<iframe>`.

Наконец 12 марта 1993 года Марк Андрессен написал в той же ветви дискуссии:

> Вернусь к теме встроенных изображений. Приближается выпуск Mosaic v0.10, в котором будет оговоренная ранее поддержка растровых изображений форматов GIF и XBM в тексте. [...]
> Поддерживать теги INCLUDE/EMBED мы в настоящее время пока не готовы. [...] Вероятно, сейчас придется остановиться на <IMG SRC="url"> (а не ICON, потому что не всякое изображение, вставленное в текст, можно назвать значком). Пока что встроенные изображения не типизируются явным образом; мы намерены начать поддержку графических типов впоследствии, когда речь зайдет о реализации системы MIME-типов в целом. Используемые нами сейчас алгоритмы чтения изображений определяют формат на лету, так что даже расширение файла не играет никакой роли.

Не отрывая карандаша

Со времен этой заочной беседы, в результате которой появился очень популярный общеупотребительный HTML-элемент, прошло почти 18 лет, но меня не перестают удивлять все детали ситуации. Судите сами.

- До сих пор существует HTTP. Успешно развиваясь, он прошел через версии 0.9, 1.0, позднее 1.1, и его эволюция на этом не останавливается (http://www.ietf.org/dyn/wg/charter/httpbis-charter.html).
- До сих пор существует HTML. Этот второстепенный формат данных, в котором сначала не поддерживались даже значки (!), позднее представал в версиях 2.0, 3.2 и 4.0. Историю HTML, если угодно, можно нарисовать на листе бумаги, не отрывая карандаша. Получится кривая и путаная линия с самопересечениями, «эволюционное древо» с многочисленными тупиковыми ветвями, отражающими моменты, когда мысль разработчика стандартов далеко опережала практику программирования и верстки. Но и в современных (2011 года) браузерах веб-страницы, созданные в 1990 году, отображаются корректно. Недавно со своего новенького мобильного аппарата на платформе Android я открыл в браузере одну из таких страниц. Система даже не попросила меня подождать, «пока будет загружен устаревший формат».
- Лицо стандартов HTML всегда определяли не только собственно стандартизаторы, но и программисты (разработчики браузеров), и веб-мастера (создатели веб-страниц), и множество непрофессиональных любителей поговорить о гипертекстовой разметке. Успешные версии HTML были, как правило, ретроспективными и описывали существующее положение вещей, вместе с тем пытаясь направить развитие в нужное русло. Если кто-то будет доказывать вам, что надо соблюдать «чистоту» стандарта HTML (то есть, по сути, что надо игнорировать разработчиков браузеров, или веб-мастеров, или тех и других), знайте: такой человек просто некомпетентен. Никогда HTML не был чистым, и все попытки «очистить» его заканчивались провалом тех же масштабов, что и попытки чем-нибудь заменить его.
- Ушли со сцены не только те браузеры, которыми пользовались в 1993 году, но и все их потомки. Netscape Navigator прекратил свое существование в 1998 году. Будучи переписан с нуля, он вошел в состав Mozilla Suite, ответвлением в развитии которого стал современный Firefox. Робкий первый шаг истории Internet Explorer можно усмотреть в браузере из пакета дополнений Microsoft Plus! к операционной системе Windows 95, где содержались также игра Пинбол и темы Рабочего стола.
- До сих пор сохранились некоторые операционные системы из числа тех, которыми пользовались в 1993 году, но ни одной из них не принадлежит сколько-нибудь значительная доля пользователей сети. Сейчас выходят в Интернет с персональных компьютеров из-под Windows версии 2000 или более поздней, с Macintosh из-под Mac OS X, с компьютеров из-под разных дистрибутивов Linux, а также с устройств карманного формата вроде iPhone. Вспомним, что в 1993 году версия 3.1 системы Windows сражалась за рынок с OS/2, на Macintosh

работала System 7, а Linux распространялся среди пользователей Usenet. Это времена Trumpet Winsock и MacPPP, по которым испытывают неудержимую ностальгию старые интернет-пользователи.

- К тому, что сейчас называется попросту веб-стандартами, до сих пор (почти через 20 лет!) имеют отношение некоторые из участников той беседы 1993 года. Иные из них еще в 1980-е годы и даже ранее работали над более старыми проектами, родственными HTML.
- Поскольку зашла речь о проектах-предшественниках, то отмечу, что в эпоху всеобщей популярности HTML и Интернета другие (мощно повлиявшие на них) форматы и системы рубежа 1980–1990-х годов оказались прочно забытыми. Скажите, приходилось ли вам раньше слышать о проектах Andrew и Intermedia? Или о системе HyTime? А ведь HyTime — это не какая-нибудь недолговечная академическая разработка. Она была ISO-стандартизована для военных нужд, с ней связывали большие ожидания, и вокруг нее крутились большие деньги, как вы можете прочесть в статье http://www.sgmlsource.com/history/hthist.htm.

Однако все эти теоретические рассуждения не дают ответа на самый первый наш вопрос: откуда взялся тег `<img>`? Почему не суждено было возникнуть тегу `<icon>` или `<include>`? Почему мы не пользуемся гиперссылками с атрибутом `include` или каким-либо сочетанием ссылочных отношений? Что обусловило решающий перевес именно в пользу `<img>`? Ответ прост: Марк Андрессен выпустил в свет программу, где была реализована поддержка именно `<img>`, а выпуск решает дело.

Нельзя сказать, что так происходит *всегда*. Ведь создатели Andrew, Intermedia и HyTime тоже выпускали приложения в свет. Работающий опубликованный код — это необходимое, но *не достаточное* условие успеха. Разумеется, я не хочу сказать, что выпуск программной реализации до утверждения стандарта решает все проблемы. Напротив, введенный Марком тег `<img>` не оговаривал единого формата графики; никак не было определено обтекание картинки текстом; не поддерживались текстовые альтернативы и заместительное содержимое для более старых браузеров. Сейчас, 18 лет спустя, мы все еще сражаемся с контент-сниффингом, который продолжает служить источником большого количества уязвимостей. Историю борьбы со всеми этими проблемами можно проследить от наших дней через эпоху «великих войн браузеров» вплоть до того самого 25 февраля 1993 года, когда Марк Андрессен мимоходом заметил: «Может быть, когда-нибудь с помощью MIME-типов...» — и решил все-таки выпустить свою реализацию.

Основные вехи развития HTML в 1997–2004 годах

В декабре 1997 года Консорциум Всемирной паутины (W3C) опубликовал спецификацию HTML 4.0, и почти сразу же полномочия рабочей группы, ответственной за выработку стандарта HTML, были прекращены. Менее чем через два месяца после этого другая рабочая группа Консорциума опубликовала стандарт XML 1.0.

Прошло еще месяца три, и по инициативе W3C был проведен семинар «Контуры будущего HTML», от участников которого ожидали ответа на вопрос: «Должен ли W3C остановить работу над HTML?» Участники дискуссии сошлись на таком вердикте:

> Обсуждение показало, что было бы чересчур трудоемкой процедурой как расширять HTML 4.0, так и превращать его в XML-приложение. Чтобы освободиться от сдерживающих факторов, предлагаем начать разработку нового поколения стандарта HTML на основе набора XML-тегов.

Для создания такого набора XML-тегов W3C вновь созвал рабочую группу HTML. В первую очередь в декабре 1998 года члены группы написали промежуточную спецификацию, которая переформулировала существующие элементы и атрибуты HTML в терминах XML; никаких новых элементов и атрибутов добавлено не было. Эта спецификация позднее стала известна как XHTML 1.0. Она определяла новый MIME-тип для документов XHTML: `application/xhtml+xml`. Упростить для уже существующих страниц (HTML4) переход на новый стандарт было призвано приложение C, в котором кратко излагались «указания по дизайну для веб-мастеров, которые хотели бы, чтобы их XHTML-документы корректно отображались в пользовательских агентах, существующих для стандарта HTML». Приложение C гласит, что можно создавать страницы, условно называемые *XHTML-страницами*, и оснащать их MIME-типом `text/html`.

Следующим пунктом на повестке дня для рабочей группы HTML были веб-формы. В августе 1999 года члены группы опубликовали первый вариант стандарта XHTML Extended Forms. В самых первых строках этого документа (http://www.w3.org/TR/1999/WD-xhtml-forms-req-19990830#intro) сказано, какие ожидания на него возлагаются.

При внимательном рассмотрении рабочая группа HTML нашла, что к задачам нового поколения веб-форм не может относиться обратная совместимость с браузерами, предназначенными для отображения документов более ранних версий HTML. Наша цель — построить ясную современную модель форм (далее расширенные формы XHTML — XHTML Extended Forms), которая была бы основана на совокупности тщательно оговоренных требований. Требования, которые описаны далее в этом документе, своей отправной точкой имеют опыт эксплуатации самых разнообразных приложений на формах.

Еще через несколько месяцев документ XHTML Extended Forms был переименован в XForms и к его доработке приступила специально сформированная рабочая группа. Она вела свои обсуждения параллельно с рабочей группой HTML и опубликовала первую официальную версию стандарта XForms 1.0 в октябре 2003 года.

После того как переход на XML был завершен, члены рабочей группы HTML задумались о том, чтобы наконец создать «новое поколение стандарта». В мае 2001 года вышло первое издание XHTML 1.1, где поверх XHTML 1.0 было добавлено совсем немного функциональности и перестала существовать прежняя лазейка для веб-мастеров, оговоренная в приложении C. Все документы XHTML, начиная с версии 1.1, оснащаются MIME-типом `application/xhtml+xml`.

Все, что вы знаете об XHTML, — это ложь

Почему так важны MIME-типы? Почему я не устаю возвращаться к ним? Причиной тому «драконовская» обработка ошибок.

Браузеры всегда очень мягко обходились с ошибками в HTML-коде. Если, например, вы создали HTML-страницу без тега `<title>`, то браузер все равно отобразит ее, несмотря на то что всегда, во всех версиях стандарта HTML присутствие элемента `<title>` на странице требовалось безоговорочно. Некоторые теги нельзя вкладывать в другие теги, но, если вы создадите страницу, нарушающую это требование, браузер все равно каким-то одному ему ведомым способом сумеет отобразить ее, не выводя сообщения об ошибке.

Как нетрудно догадаться, тот факт, что «кривая» HTML-разметка все же отображается в браузерах, позволил беззаботным веб-программистам создать много «кривых» страниц. Очень много. По некоторым оценкам, сейчас в Сети свыше 99 % всех HTML-страниц содержат не менее одной ошибки верстки каждая. Но при таких ошибках браузеры не выводят предупреждения — вот почему «кривой» HTML-код никто никогда не правит.

Видя в этом одну из фундаментальных проблем современного Интернета, специалисты W3C задумались о ее решении. Опубликованный в 1997 году стандарт XML нарушил традицию мягкой обработки ошибок в клиентских программах. Согласно этому стандарту, все программы отображения и обработки XML должны воспринимать ошибки структуры документа как фатальные. Система, при которой первая же ошибка вызывает сбой, стала известна как «драконовская» обработка ошибок, названная так по имени афинского вождя Дракона (в иных источниках: Драконта). Введенные им законы предписывали карать смертью даже незначительные проступки. После того как Консорциум переформулировал HTML в терминах XML-словаря, ответственные лица постановили, чтобы обработка ошибок в документах нового MIME-типа `application/xhtml+xml` была «драконовской». Иными словами, если бы в код вашей XHTML-страницы закралась одна-единственная неточность, браузеру пришлось бы остановить работу и показать пользователю (посетителю страницы) сообщение об ошибке.

Эту идею приняли не все. Как уже говорилось, 99 % веб-страниц не свободно от ошибок и поэтому при использовании `application/xhtml+xml` есть высокие шансы, что конечный пользователь не увидит страницы. И хотя новая функциональность XHTML 1.0 и 1.1 все же очень привлекательна, веб-программисты преимущественно избегали использовать тип `application/xhtml+xml`. Это, разумеется, не значит, что они игнорировали также и весь стандарт XHTML. Совсем наоборот. Приложение C спецификации XHTML 1.0 предоставляло ту самую лазейку, о которой уже сказано выше. По сути, веб-мастеру говорили: «Пользуйся синтаксисом в духе XHTML, но оснащай страницу MIME-типом `text/html`». Многие разработчики так и поступали: фактически перейдя на XHTML-синтаксис, они по-прежнему использовали тип `text/html`.

В наши дни в коде очень многих веб-страниц в первой строке объявлен тип документа XHTML, имена тегов набраны строчными буквами, значения атрибутов взяты в кавычки, а при оформлении одиночных тегов применяется косая чер-

та, например `<br />`, `<hr />`. Но MIME-тип `application/xhtml+xml`, включающий «драконовскую» обработку ошибок по правилам XML, определяется лишь в ничтожном меньшинстве таких страниц. Любую страницу MIME-типа `text/html`, невзирая на объявленный тип документа, синтаксис и стиль, разбирает HTML-парсер, очень лояльный к ошибкам. Он молча терпит некорректную разметку, не уведомляя о ней ни конечного пользователя, ни вообще кого-либо.

Итак, лазейка имелась в XHTML 1.0, а в XHTML 1.1 ее закрыли. Неоконченный набросок спецификации XHTML 2.0 тоже предусматривает последовательную «драконовскую» обработку ошибок. Вот почему о своем соответствии стандарту XHTML 1.0 заявляют много миллиардов страниц, тогда как на разбор по правилам XHTML 1.1 или XHTML 2.0 претендует очень-очень мало веб-документов. Теперь задумайтесь: действительно ли вы пользуетесь XHTML? Ответить поможет MIME-тип (если вам вообще неизвестно, каким MIME-типом вы пользуетесь, могу почти наверняка сказать, что это по старинке `text/html`). Ваш так называемый XHTML будет XML-форматом лишь по названию все то время, пока вашим веб-страницам не задается тип `application/xhtml+xml`.

Альтернативная точка зрения

В июне 2004 года W3C провел семинар по веб-приложениям и сложным документам. На семинаре присутствовали представители фирм, производящих браузеры, компаний, занимающихся веб-разработкой, и других организаций — членов W3C. Во время мероприятия группа заинтересованных сторон (в числе которых были Mozilla Foundation и Opera Software) презентовала свой альтернативный взгляд на будущее Интернета: эволюция стандарта HTML4 путем включения в него новых функций для нужд разработчиков современных веб-приложений (http://www.w3.org/2004/04/webapps-cdf-ws/papers/opera.html).

Важнейшие требования к работам в этом направлении могут быть, на наш взгляд, выражены следующими семью принципами.

Обратная совместимость, ясный путь преобразований

Веб-приложения должны быть основаны на известных веб-мастерам технологиях, таких как HTML, CSS, DOM и JavaScript. Базовая функциональность веб-приложений должна быть реализуемой в наши дни средствами IE6, чтобы путь дальнейших преобразований был ясен веб-мастерам. Крайне маловероятен успех какого-либо решения, которое не поддерживают (без установки исполняемых приложений) пользовательские агенты, доминирующие сейчас на рынке.

Хорошо продуманная обработка ошибок

Обработка ошибок в веб-приложениях должна быть настолько детализирована, чтобы пользовательским агентам не приходилось в этих целях создавать собственные механизмы обработки ошибок или проводить обратную разработку [соответствующих механизмов] других пользовательских агентов.

Ошибки верстки не следует показывать пользователю

Для каждого из возможных сценариев ошибки спецификация должна указывать точный способ исправления ошибки. В обработке большей части ошибок нужно «мягкое» исправление, как в CSS, а не катастрофическая остановка работы, как в XML.

Практическая значимость

Каждую функцию, вносимую в стандарт веб-приложений, должен оправдывать случай использования на практике. Обратное, вообще говоря, неверно: не каждый пример практического

использования подтверждает значимость функции. Самыми удачными следует считать примеры с действующих сайтов, на которых для обхода ограничений ранее использовался неудачный путь.

Сценарии надо сохранить

Но там, где более удобна декларативная разметка, ими не следует злоупотреблять. Сценарии должны быть нейтральны к типу устройства и способу представления, кроме тех, что работают на конкретных устройствах (например, внесенных в базу XBL).

Следует избегать профилирования под конкретное устройство

В десктопной и мобильной версиях одного и того же пользовательского агента веб-мастерам должен быть предоставлен один и тот же набор функций.

Открытый процесс

Разработка в открытой среде пошла на пользу современному Интернету. Поскольку веб-приложения — это ядро будущего Всемирной паутины, их разработку тоже следует сделать открытой. Доступ к дискуссионным листам, архивам и черновикам спецификаций должен быть все время открыт для посетителей.

Было проведено неофициальное голосование, в рамках которого участников семинара попросили ответить на вопрос: «Должен ли W3C разрабатывать декларативные расширения HTML, CSS и императивные расширения DOM для обслуживания потребностей веб-приложений среднего уровня (то есть расширения, уступающие по своему масштабу полноценным API уровня операционных систем)?» Было получено 8 голосов «за» и 11 — «против». Подводя итоги работы семинара (http://www.w3.org/2004/04/webapps-cdf-ws/summary), члены W3C писали: «В настоящее время W3C не намеревается вкладывать средства в третью из тем, вынесенных на неофициальное голосование: расширения HTML и CSS для нужд веб-приложений. Это не касается технологий, уже разрабатываемых в настоящее время рабочими группами W3C».

Перед теми, кто предлагал развивать далее HTML и веб-формы, открылось два пути: или махнуть на все рукой, или продолжить свою деятельность вне W3C. Был выбран второй вариант, и в июне 2004 года, вслед за регистрацией доменного имени whatwg.org, появилась рабочая группа WHAT.

WHAT? Что?!

Что это вообще за группа — WHAT? Предоставлю слово им самим (http://www.whatwg.org/news/start).

Рабочая группа по вопросам технологии гипертекстовых веб-приложений (Web Hypertext Applications Technology) — это форма свободного, неофициального, открытого сотрудничества между разработчиками браузеров и любыми заинтересованными лицами. Группа видит свою цель в том, чтобы на основе HTML и родственных технологий выработать спецификации, которые облегчили бы развертывание совместимых веб-приложений. Предполагается, что результаты деятельности группы будут переданы стандартизующей организации и послужат основой для формального расширения HTML в области стандартов.

Созданию этого форума предшествовало несколько месяцев частной переписки о спецификациях веб-технологий. Так сложилось, что в настоящее время мы заняты в основном дополнением HTML4-форм востребованными функциями без нарушения обратной совместимости. Группа была создана с тем, чтобы дальнейшее развитие этой и других спецификаций стало совершенно открытым и велось на дискуссионном листе с общедоступным архивом.

«Без нарушения обратной совместимости» — ключевые слова этого пассажа. XHTML (за вычетом той возможности, которую открывает Приложение С) не является, вообще говоря, обратно совместимым с HTML и требует отдельного MIME-типа. Все содержимое, помеченное этим типом, будет подлежать «драконовскому» механизму обработки ошибок. XForms также не обладает обратной совместимостью с HTML-формами, потому что формы XForms можно использовать только в документах с новым MIME-типом (и, следовательно, «драконовской» обработкой ошибок). Все дороги ведут к MIME-типизации.

Рабочая группа WHAT не стала задаваться вопросом, принесли ли какую-нибудь пользу десять лет инвестиций в HTML. Не стала она и нарушать работу 99 % страниц Интернета. Участники группы пошли другим путем — путем документирования алгоритмов «мягкой» обработки ошибок в современных браузерах. Браузеры всегда «прощали» ошибки в HTML-коде, но почему-то никому не приходило в голову подробно описать, как это происходит. И вот Netscape пытался имитировать обработку «кривых» страниц теми алгоритмами, которые были в NCSA Mosaic. Потом Internet Explorer, в свою очередь, пытался имитировать Netscape; затем Opera и Firefox пытались имитировать Internet Explorer; потом Safari пытался имитировать Firefox и т. д. вплоть до настоящего времени. У разработчиков ушло много тысяч часов только на то, чтобы обеспечивать в этой части единообразие работы своих и конкурирующих продуктов.

Этот огромный труд, к счастью, уже закончился. За несколько лет рабочая группа WHAT смогла успешно документировать (за вычетом нескольких неясных предельных случаев) алгоритм разбора HTML, совместимый с существующим веб-содержимым. Нигде в этом алгоритме нет такого шага, как остановка работы пользовательского агента и вывод сообщения об ошибке. Одновременно с этой процедурой обратной разработки группа WHAT занималась и множеством других вещей. В их числе была спецификация, первоначально названная Web Forms 2.0, которая предусматривала новые типы элементов управления в веб-формах (больше о веб-формах вы прочтете в главе 9). Еще один подобный черновой документ — Web Applications 1.0 — был посвящен веб-приложениям и предусматривал много новых функций, таких как непосредственный интерфейс рисования — холст (см. главу 4) и встроенная, без специальных приложений, поддержка видео и аудио (см. главу 5).

И снова о W3C

На протяжении нескольких лет W3C и рабочая группа WHAT совершенно не интересовались друг другом. Специалисты WHAT сосредоточились на веб-формах и новых функциях HTML, а созванная W3C рабочая группа HTML была занята разработкой версии 2.0 стандарта XHTML. Впрочем, к октябрю 2006 года стало ясно, что группа WHAT обрела широкую поддержку, в то время как XHTML 2 все еще томился в черновиках и не был реализован ни в одном популярном браузере. Тогда Тим Бернерс-Ли, основатель Консорциума Всемирной паутины W3C, объявил о начале совместного труда W3C и группы WHAT над развитием HTML (http://dig.csail.mit.edu/breadcrumbs/node/166).

> Спустя несколько лет некоторые факты становятся яснее. HTML надо развивать постепенно. Попытка сразу переучить всех на XML с его обязательными кавычками вокруг значений атрибутов, обратными слешами в одиночных тегах и пространствами имен не удалась. HTML-верстальщики в своем большинстве не изменили навыков во многом потому, что этому потворствовали браузеры. Есть большие сообщества, которые перешли на новый стандарт и теперь наслаждаются благами правильной системы, но таковы не все. Следует дорабатывать HTML шаг за шагом, чтобы тем самым постепенно делать мир правильнее, гармоничнее, получать власть над гармонией этого мира.
>
> Наш план — созвать совершенно новую группу HTML. В отличие от предшественников, ее сотрудники будут уполномочены последовательно улучшать HTML и вместе с тем XHTML. У этой группы будет другой председатель и другой контактный адрес. Работа над HTML и XHTML будет вестись совместно. Мы уже заручились поддержкой этого начинания со стороны многих лиц, в том числе разработчиков браузеров.
>
> Ожидается также работа над веб-формами. Эта область проблематична ввиду того, что существуют и HTML-формы, и XForms. HTML-формы распространены повсеместно, но у XForms также немало реализаций и пользователей. Между тем большие дополнения к HTML-формам предлагает проект спецификации Web Forms. В связи с этим планируется расширять HTML-формы.

Одним из первых пунктов на повестке дня вновь созванной W3C рабочей группы HTML было переименовать Web Applications 1.0 в HTML5. Вот мы и пришли сюда, готовые к постижению глубин HTML5.

Послесловие

В октябре 2009 года Консорциум прекратил деятельность рабочей группы стандарта XHTML 2. Решение было объяснено так (http://www.w3.org/2009/06/xhtml-faq.html):

> Когда в марте 2007 года W3C объявил о начале деятельности рабочих групп HTML и XHTML 2, было отмечено, что Консорциум продолжит наблюдать за потенциальным рынком XHTML 2. W3C сознает, как важно дать сообществу разработчиков недвусмысленный знак относительно будущего HTML. По этой причине, признавая ценность многолетнего труда рабочей группы XHTML 2, администрация W3C после совещания с участниками Консорциума решила ограничить полномочия данной рабочей группы 2009 годом и далее их не продлевать.

Итак, кто выпускает программы, тот и заказывает музыку.

Для дальнейшего изучения

- «История Сети» (http://hixie.ch/commentary/web/history) — неоконченная старая статья Яна Хиксона (Ian Hickson).
- «HTML/История» (http://www.w3.org/html/wg/wiki/History) — публикация Майкла Смита (Michael Smith), Генри Сивонена (Henri Sivonen) и нескольких соавторов.
- «Краткая история HTML» (http://atendesigngroup.com/blog/brief-history-html) — статья Скотта Рейнена (Scott Reynen).

2 Тестирование функций HTML5 в браузере

Приступим

Вы могли бы, конечно, спросить: «Как же мне начинать пользоваться HTML5, если старые браузеры его не поддерживают?» Но сама постановка такого вопроса ошибочна. HTML5 — это не единый большой инструмент, а совокупность отдельных функций, поэтому «поддержку HTML5» вообще нельзя протестировать: это выражение бессмысленно. Но можно проверить, поддерживает ли браузер отдельные нововведения: холст, видео, геолокацию и др.

Способы тестирования браузера

При прорисовке страницы браузер строит объектную модель документа (DOM) — набор объектов, которыми представляются HTML-элементы страницы. Каждый тег — `<p>`, `<div>`, `<span>` и т. д. — представляется в DOM-структуре отдельным объектом (есть также глобальные объекты — окно и целый документ, не привязанные к специфическим HTML-элементам).

У всех DOM-объектов есть общие свойства, но у некоторых объектов их больше, чем у других. Более того, в браузерах, поддерживающих HTML5-функциональность, какие-то объекты будут располагать уникальными свойствами. Поддерживается ли та или иная функция, можно увидеть, заглянув в DOM.

Существуют четыре основных способа тестирования браузера на предмет поддержки разных возможностей HTML5. Рассмотрим их по порядку, от простейшего к более сложным.

1. Проверить, определено ли нужное свойство для такого глобального объекта, как `window` или `navigator`.

   Пример — поддержка геолокации. Как протестировать браузер на ее наличие, читайте в разделе «Геолокация» данной главы.

2. Создать элемент и проверить, определено ли для него нужное свойство.

   Пример — поддержка рисования. О ней вы узнаете в разделе «Холст» этой главы.

3. Создать элемент, проверить, определен ли для него нужный метод, затем вызвать этот метод и посмотреть на возвращенное им значение.

Пример — поддержка видеоданных разных форматов. О ней читайте в разделе «Форматы видео» этой главы.

4. Создать элемент, установить для него нужное значение какого-либо свойства, затем проверить, сохраняет ли свойство данное значение.

   Пример — поддержка разных типов полей ввода. О ней вы узнаете в разделе «Типы полей ввода» данной главы.

Modernizr: библиотека для тестирования HTML5-функций

Modernizr (http://www.modernizr.com) — это JavaScript-библиотека, распространяемая под лицензией MIT с открытым исходным кодом. Для большинства возможностей HTML5 и CSS3 она предоставляет простой способ проверить, поддерживает ли их исследуемый браузер. На момент написания этой книги последняя версия Modernizr имела номер 1.1[1]. Обязательно применяйте самую новую версию. Для использования Modernizr надо включить в головную часть страницы следующий тег <script>:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Название моей страницы</title>
  <script src="modernizr.min.js"></script>
</head>
<body>
  ...
</body>
</html>
```

Modernizr запускается автоматически, поэтому нет необходимости в функции modernizr_init(), которая бы вызывала ее. При запуске библиотека Modernizr создает одноименный объект с набором булевозначных свойств, по одному на каждую распознаваемую функцию. Так, например, если ваш браузер поддерживает API рисования (см. главу 4), то свойство Modernizr.canvas примет значение true, а если нет — false:

```
if (Modernizr.canvas) {
  // порисуем!
} else {
  // тег <canvas> не поддерживается, печаль :(
}
```

Холст

В HTML5 тег <canvas> (http://bit.ly/9JHzOf) определен как «холст для зависимой от разрешения растровой графики, с помощью которого могут динамически прори-

[1] В январе 2011 года были доступны стабильная версия 1.6 и бета-версия 2.0. — *Примеч. перев.*

совываться диаграммы, графика игр и прочие изображения». На странице холст имеет вид прямоугольника, в границах которого можно рисовать с помощью JavaScript. В HTML5 определен «API рисования» — набор функций, позволяющих рисовать разные фигуры и контуры, создавать градиентную заливку, преобразовывать графику.

Протестировать поддержку API рисования можно способом 2 (см. раздел «Способы тестирования браузера» этой главы). Если <canvas> поддерживается в браузере, то для DOM-объекта, представляющего тег <canvas>, будет определен метод getContext() (см. раздел «Простые фигуры» главы 4). Если же поддержки рисования в браузере нет, то созданный объект не будет иметь характеристик холста. Поддерживается ли <canvas>, можно проверить с помощью такой функции:

```
function supports_canvas() {
  return !!document.createElement('canvas').getContext;
}
```

В единственной ее строке сначала создается пустой тег <canvas>:

```
return !!document.createElement('canvas').getContext;
```

Он не связан с какой-либо областью страницы и не будет отображаться на экране. Ничего не делая, он просто тихо покачивается на волнах оперативной памяти.

После создания пустого тега <canvas> надо проверить, определен ли для него метод getContext(). Этот метод определен только в том случае, если браузер поддерживает API рисования:

```
return !!document.createElement('canvas').getContext;
```

Наконец, двойное отрицание заставляет систему вывести значение булевого типа (true или false).

```
return !!document.createElement('canvas').getContext;
```

Данным способом можно исследовать, поддерживается ли в браузере большинство функций API рисования: фигуры (см. раздел «Простые фигуры» главы 4), контуры (см. раздел «Контуры» главы 4), градиенты (см. раздел «Градиенты» главы 4) и узоры. Таким образом, не может быть обнаружена поддержка сторонней библиотеки ExplorerCanvas (см. раздел «А что в IE?» главы 4), реализующей API рисования в Internet Explorer.

Чтобы не писать собственную функцию для проверки того, поддерживается ли в браузере API рисования, прибегните к помощи Modernizr (см. предыдущий раздел):

```
if (Modernizr.canvas) {
  // порисуем!
} else {
  // тег <canvas> не поддерживается :(
}
```

API рисования текста нужно тестировать особо, о чем и пойдет речь далее.

Рисование текста

Если даже ваш браузер поддерживает тег <canvas> и API рисования, в нем может не быть поддержки API рисования текста. Инвентарь функций рисования расширялся постепенно, текстовые функции были добавлены сравнительно поздно, и некоторые браузеры с поддержкой рисования вышли в свет прежде, чем разработка текстового API была завершена.

Протестировать поддержку API рисования текста вновь позволяет способ 2 (см. раздел «Способы тестирования браузера» этой главы). Если рисование текста в вашем браузере поддерживается, то для DOM-объекта, представляющего тег <canvas>, будет определен метод getContext() (см. раздел «Простые фигуры» главы 4), а если нет, то соответствующий DOM-объект не будет располагать свойствами, специфичными для холста. Проверить, работает ли в браузере рисование текста, вам поможет такая функция:

```
function supports_canvas_text() {
  if (!supports_canvas()) { return false; }
  var dummy_canvas = document.createElement('canvas');
  var context = dummy_canvas.getContext('2d');
  return typeof context.fillText == 'function';
}
```

В начале кода этой функции вызывается ранее написанная нами функция supports_canvas(), которая тестирует поддержку API рисования:

```
if (!supports_canvas()) { return false; }
```

Понятно, что, если браузером не поддерживается тег <canvas>, то и рисование текста в нем будет невозможно.

Теперь создадим пустой тег <canvas> и выясним его контекст рисования. Это непременно удастся сделать, так как функция supports_canvas() уже удостоверилась в том, что метод getContext() определен для всех объектов-холстов:

```
var dummy_canvas = document.createElement('canvas');
var context = dummy_canvas.getContext('2d');
```

Выясним, наконец, существует ли в контексте рисования функция fillText(). Если да, то API рисования текстом доступен:

```
return typeof context.fillText == 'function';
```

Чтобы не писать собственную функцию, прибегните к помощи Modernizr:

```
if (Modernizr.canvastext) {
  // порисуем текст!
} else {
  // порисовать текст не удастся :(
}
```

Видео

В HTML5 появился новый тег <video>, предназначенный для встраивания видеофрагментов в веб-страницы. Раньше это было невозможно без сторонних приложений, таких как Apple QuickTime и Adobe Flash.

Спецификация тега <video> такова, что его можно использовать без вспомогательных сценариев. Если указать несколько видеофайлов одного и того же содержания, то браузер с поддержкой HTML5-видео сам выберет, исходя из поддерживаемых форматов, какой из них показать пользователю[1].

Браузеры, которые не поддерживают HTML5-видео, совершенно игнорируют теги <video>. Но этот факт можно обратить в свою пользу и заставить браузер воспроизводить видео с помощью стороннего приложения. Такое решение предложил Крок Кэмен (Kroc Camen). Его разработка Video for Everybody! («Видео для каждого», http://camendesign.com/code/video_for_everybody) позволяет опираться на возможности HTML5-видео там, где они присутствуют, а в более старых браузерах пользоваться QuickTime или Flash. Разработка Кэмена построена не на основе JavaScript и поэтому может обслуживать, в принципе, любой браузер, в том числе мобильный.

Чтобы выполнять с видеоданными кое-какие операции посложнее, чем только вставка на страницу и воспроизведение, надо пользоваться JavaScript. Протестировать поддержку видео можно способом 2 (см. раздел «Способы тестирования браузера» этой главы). Если в браузере поддерживается HTML5-видео, то для DOM-объекта, соответствующего тегу <video>, будет определен метод canPlayType(), а при отсутствии поддержки HTML5-видео набор свойств DOM-объекта будет стандартным. Проверить, работает ли в браузере новая видеофункциональность, вам поможет следующая функция:

```
function supports_video() {
  return !!document.createElement('video').canPlayType;
}
```

Чтобы не писать собственную функцию, прибегните к помощи Modernizr (см. раздел «Modernizr: библиотека для тестирования HTML5-функций» этой главы):

```
if (Modernizr.video) {
  // воспроизведем ролик!
} else {
  // встроенной поддержки видео нет,
// надо пользоваться QuickTime или Flash
}
```

Отдельно нужно определять, какие форматы видео умеет воспроизводить браузер. Об этом — следующий раздел.

Форматы видео

Форматы видеоданных сродни разным языкам. В английской газете может быть написано то же самое, что и в испанской, но если вы умеете читать только по-английски, то вторая газета будет вам без надобности (во всяком случае для чтения). Так и с видео: чтобы воспроизвести ролик, браузер должен знать «язык» его данных.

---

[1] О различных форматах видео читайте «Элементарное введение в кодировки видеоданных»: часть 1 «Форматы файлов» (http://diveintomark.org/archives/2008/12/18/give-part-1-container-formats) и часть 2 «Видеокодеки, сжимающие с потерями» (http://diveintomark.org/archives/2008/12/19/give-part-2-lossy-video-codecs). — *Примеч. авт.*

«Языки» видеоформатов называются кодеками. *Кодек* — это алгоритм, в соответствии с которым последовательность графических образов кодируется последовательностью бит. Сейчас в мире несколько десятков широко используемых кодеков. Какой же выбрать? Суровая реальность HTML5 такова, что разработчики браузеров не сумели договориться о едином общепринятом видеокодеке. Впрочем, количество альтернатив немногим больше: две. Один из этих кодеков платный (лицензионный сбор), но работает в Safari и на iPhone (а также в Adobe Flash, если пользоваться разработками типа «Видео для каждого»). Другой — бесплатный, он работает в браузерах с открытым исходным кодом, таких как Chromium и Mozilla Firefox.

Протестировать поддержку разных форматов видео можно способом 3 (см. раздел «Способы тестирования браузера» этой главы). Если в браузере поддерживается HTML5-видео, то для DOM-объекта, соответствующего тегу `<video>`, будет определен метод `canPlayType()`, который и сообщит, с какими форматами видео может работать браузер.

Рассмотрим функцию, проверяющую браузер на умение читать патентованный формат Macintosh и iPhone:

```
function supports_h264_baseline_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
}
```

В первой строке этого кода с помощью созданной ранее функции `supports_video()` мы проверяем, есть ли в браузере поддержка HTML5-видео:

```
if (!supports_video()) { return false; }
```

Если браузер не поддерживает HTML5-видео, то и разные форматы он, конечно, не будет читать.

Теперь создадим пустой тег `<video>` (но не будем присоединять его к странице, чтобы он не был виден) и вызовем метод `canPlayType()`. Этот метод наверняка будет доступен, ведь в его существовании только что удостоверилась функция `supports_video()`.

```
var v = document.createElement("video");
return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
```

Формат видеоданных — это, вообще говоря, сочетание нескольких компонентов. Если говорить технически, показанный выше код спрашивает у браузера, может ли тот воспроизводить видео H.264 базового профиля и звук AAC профиля низкой сложности в контейнере MPEG-4[1].

Функция `canPlayType()` не возвращает булевы значения `true` и `false`. Зная, что некоторые видеоформаты очень сложны, функция отвечает одним из трех значений:

---

[1] Что все это значит, я объясню в главе 5. Возможно, вам не помешает также ознакомиться с «Элементарным введением в кодировки видеоданных» (http://diveintomark.org/tag/give). — *Примеч. авт.*

- "probably" — браузер уверен, что может проигрывать видео данного формата;
- "maybe" — браузер считает, что, возможно, сумел бы воспроизвести видео данного формата;
- "" (пустая строка) — браузер полагает, что наверняка не может воспроизвести такое видео.

Для проверки браузера на умение читать открытый формат, используемый в Mozilla Firefox и иных браузерах с открытым кодом, есть другая функция. Она сконструирована точно так же. Разница лишь в том, что функции canPlayType() передается не такая строка, как в примере выше. На этот раз мы спрашиваем у браузера, может ли тот воспроизводить видео формата Theora и аудио формата Vorbis в Ogg-контейнере:

```
function supports_ogg_theora_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/ogg; codecs="theora, vorbis"');
}
```

Есть еще третья альтернатива — WebM (http://www.webmproject.org), не защищенный патентом видеокодек, код которого недавно был открыт. Его намереваются включить в последующие версии популярных браузеров, в частности Chrome, Firefox и Opera. Тестирование поддержки WebM-видео проводится по той же схеме:

```
function supports_webm_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/webm; codecs="vp8, vorbis"');
}
```

Чтобы не писать собственную функцию, можете прибегнуть к помощи Modernizr (но имейте в виду, что тестировать поддержку открытого видеоформата WebM эта библиотека пока не умеет):

```
if (Modernizr.video) {
  // воспроизведем ролик, но вот в каком формате?
  if (Modernizr.video.ogg) {
    // попробуем Ogg Theora + Vorbis в Ogg-контейнере
  } else if (Modernizr.video.h264){
    // попробуем видео H.264 + аудио AAC в MP4-контейнере
  }
}
```

Локальное хранилище

Локальное хранилище HTML5 (http://dev.w3.org/html5/webstorage/) дает способ хранить часть сайтовых данных в памяти вашего компьютера, чтобы потом было удобнее их загружать. На схожей идее основана система cookies, но мы говорим о значительных

объемах информации. Cookies имеют ограниченный размер, и каждый раз при загрузке очередной страницы ваш браузер отсылает их веб-серверу, а это лишний расход времени и трафика. Локальное хранилище HTML5 устроено так, что при загрузке сайтовой страницы недостающие данные могут быть получены из памяти вашего компьютера с помощью JavaScript.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Локальное хранилище — это действительно часть HTML5? И почему ему посвящена особая спецификация?

Ответ: Если кратко, то да: локальное хранилище — часть HTML5. Чуть более подробный ответ таков: исторически локальному хранилищу был посвящен раздел в основной спецификации HTML5, но потом некоторые участники рабочей группы HTML5 пожаловались на чрезмерно возросший объем стандарта, и спецификацию хранилища выделили в особый документ. Поступать так, конечно, ничуть не разумнее, чем разрезать торт на кусочки, чтобы уменьшить общее количество калорий в нем.

Протестировать поддержку локального хранилища HTML5 можно способом 1 (см. раздел «Способы тестирования браузера» этой главы). Если HTML5-хранилище поддерживается браузером, то для глобального объекта `window` будет определено свойство `localStorage`. Соответственно, если поддержки хранилища в браузере нет, то свойство тоже не будет определено. Проверить поможет такая функция:

```
function supports_local_storage() {
  return ('localStorage' in window) && window['localStorage'] !== null;
}
```

Чтобы не писать собственную функцию, прибегните к помощи Modernizr:

```
if (Modernizr.localstorage) {
  // хранилище доступно!
} else {
  // встроенной поддержки хранилища нет,
  // стоит попробовать Gears или иное стороннее решение
}
```

Помните, что JavaScript чувствителен к регистру. Атрибут объекта Modernizr называется `localstorage` (все буквы строчные), а DOM-свойство — `window.localStorage` (буква S прописная).

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Насколько хорошо защищены данные моего локального хранилища HTML5? Может ли кто-нибудь их читать?

Ответ: Читать данные вашего локального хранилища и даже видоизменять их могут все лица, имеющие физический доступ к вашему компьютеру. Через браузер к локальному хранилищу могут обращаться все сайты, но каждому из них разрешено читать и видоизменять только «свои» данные, а не информацию, сохраненную другими сайтами. Это так называемое «ограничение по источнику» (http://bit.ly/9YyPpj).

Фоновые вычисления

Фоновые вычисления (http://bit.ly/9jheof) — введенный в HTML5 стандартный способ запуска JavaScript в браузере в фоновом режиме. Механизм фоновых вычислений позволяет иметь несколько «процессов», процедуры которых выполняются более или менее одновременно (чтобы понять, как это происходит, подумайте об одновременной работе большого количества приложений в вашем компьютере). Эти фоновые «потоки» могут выполнять сложные математические расчеты, запрашивать о чем-либо сеть или локальное хранилище, в то время как отображаемая в браузере страница будет реагировать на действия пользователя: щелчки кнопкой мыши, прокрутку, клавиатурный ввод.

Протестировать поддержку фоновых вычислений можно способом 1 (см. раздел «Способы тестирования браузера» этой главы). Если в вашем браузере поддерживается соответствующий интерфейс (Web Worker API), то для глобального объекта `window` будет определено свойство `Worker`. Соответственно, если поддержки API фоновых вычислений в браузере нет, то свойство тоже не будет определено. Проверить поможет такая функция:

```
function supports_web_workers() {
  return !!window.Worker;
}
```

Чтобы не писать собственную функцию, прибегните к помощи Modernizr (см. раздел «Modernizr: библиотека для тестирования HTML5-функций» этой главы):

```
if (Modernizr.webworkers) {
  // фоновые вычисления доступны!
} else {
  // встроенной поддержки фоновых вычислений нет,
  // стоит попробовать Gears или иное стороннее решение
}
```

Помните, что JavaScript чувствителен к регистру. Атрибут объекта Modernizr называется `webworkers` (все буквы строчные), а DOM-объект — `window.Worker` (буква W прописная).

Офлайновые веб-приложения

Читать статические веб-страницы, отключившись от Сети, очень просто: достаточно зайти в Интернет, загрузить страницу, потом выйти из Интернета, уехать с ноутбуком к себе на дачу и там в тишине наслаждаться чтением. (Шутка. На дачу можно не ехать.) С офлайновым режимом у таких веб-приложений, как Gmail и Google Docs, дело обстояло сложнее. Но теперь благодаря возможностям HTML5 создать веб-приложение, работающее в режиме офлайн, может каждый из нас, а не только компания Google.

Такое приложение берет начало в онлайне. При первом посещении сайта, который может отображаться локально, веб-сервер сообщит вашему браузеру, какие

файлы надо загрузить, чтобы сайт работал без подключения к Сети. Это могут быть абсолютно любые файлы: HTML, JavaScript, картинки и даже видеоролики (см. раздел «Видео» этой главы). После того как браузер загрузит все необходимые файлы, вы сможете открывать тот же сайт, отключившись от Интернета: браузер, видя, что сетевое подключение недоступно, будет пользоваться предварительно загруженными файлами. Перейдя в режим онлайн, вы сумеете передать на сервер измененные вами данные.

Протестировать поддержку офлайновых приложений можно способом 1 (см. раздел «Способы тестирования браузера» этой главы). Если ваш браузер поддерживает их, то для глобального объекта `window` будет определено свойство `applicationCache`. Соответственно, если поддержки офлайновых приложений в браузере нет, то свойство тоже не будет определено. Проверить это поможет такая функция:

```
function supports_offline() {
  return !!window.applicationCache;
}
```

Чтобы не писать собственную функцию, прибегните к помощи Modernizr:

```
if (Modernizr.applicationcache) {
  // офлайновые приложения доступны!
} else {
  // встроенной поддержки офлайновых приложений нет,
  // стоит попробовать Gears или иное стороннее решение
}
```

Помните, что JavaScript чувствителен к регистру. Атрибут объекта Modernizr называется `applicationcache` (все буквы строчные), а DOM-объект — `window.applicationCache` (буква C прописная).

Геолокация

Геолокация — это вид деятельности, позволяющий определять координаты своего местонахождения в мире и при желании делиться ими с друзьями. Координаты можно определить несколькими способами: по IP-адресу, через подключение к беспроводному Интернету, через ретранслятор сотовой сети, на связи с которым находится ваш телефон, или с помощью особого GPS-аппарата, принимающего спутниковые данные о широте и долготе.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Разве геолокация — это часть HTML5? Почему о ней идет речь в этой книге?

Ответ: Разработчики браузеров работают над поддержкой геолокации буквально в эти самые дни. Строго говоря, стандарт геолокации определяет самостоятельная рабочая группа (http://www.w3.org/2008/geolocation/), никак не связанная с рабочей группой HTML5. Но на страницах нашей книги упоминание о геолокации неизбежно: это один из векторов развития современного Интернета.

Протестировать поддержку геолокации можно способом 1 (см. раздел «Способы тестирования браузера» этой главы). Если ваш браузер поддерживает API геолокации, то для глобального объекта navigator будет определено свойство geolocation. Соответственно, если поддержки геолокации в браузере нет, то свойство тоже не будет определено. Проверить поможет следующая функция:

```
function supports_geolocation() {
  return !!navigator.geolocation;
}
```

Чтобы не писать собственную функцию, прибегните к помощи Modernizr (см. раздел «Modernizr: библиотека для тестирования HTML5-функций» данной главы):

```
if (Modernizr.geolocation) {
  // определим ваши координаты!
} else {
  // встроенной поддержки геолокации нет,
  // стоит попробовать Gears или иное стороннее решение
}
```

Если «родная» поддержка API геолокации в вашем браузере отсутствует, не надо отчаиваться. Gears (http://tools.google.com/gears/) — разработанное Google браузерное расширение с открытым исходным кодом, функционирующее на платформах Windows, Mac, Linux, Windows Mobile и Android, — эмулирует в старых браузерах отдельные новые функции, о которых рассказано в этой главе. В частности, Gears предлагает API геолокации. Этот интерфейс отличается от стандартного navigator.geolocation, но служит тем же целям.

Собственные API геолокации есть также на некоторых мобильных платформах, в том числе BlackBerry, Nokia, Palm и OMTP BONDI. Как пользоваться всеми этими столь несхожими прикладными интерфейсами, будет подробно рассказано в главе 6.

Типы полей ввода

Вы знаете о веб-формах все, не так ли? Действительно, чего там сложного: создать контейнер <form>, добавить несколько полей <input type="text">, возможно, одно <input type="password"> и увенчать конструкцию великолепной кнопкой <input type="submit">.

Увы, это лишь незначительная часть всех современных функций. В HTML5 появилось больше десятка новых типов полей ввода, которые теперь можно использовать в формах.

- <input type="search"> — поисковая форма (http://bit.ly/9mQt5C).
- <input type="number"> — форма выбора числа (http://bit.ly/aPZHjD).
- <input type="range"> — ползунок (http://bit.ly/dmLiRr).
- <input type="color"> — форма выбора цвета (http://bit.ly/bwRcMO).
- <input type="tel"> — телефонный номер (http://bit.ly/amkWLq).
- <input type="url"> — интернет-адрес (http://bit.ly/cjKb3a).

- `<input type="email">` — адрес электронной почты (http://bit.ly/aaDrgS).
- `<input type="date">` — форма выбора даты (http://bit.ly/c8hL58).
- `<input type="month">` — месяц (http://bit.ly/cDgHRI).
- `<input type="week">` — неделя (http://bit.ly/bR3r58).
- `<input type="time">` — метка времени (http://bit.ly/bfMCMn).
- `<input type="datetime">` — точные дата и время в абсолютном исчислении (http://bit.ly/c46zVW).
- `<input type="datetime-local">` — местные дата и время (http://bit.ly/aziNkE).

Протестировать поддержку новых типов полей ввода можно способом 4 (см. раздел «Способы тестирования браузера» этой главы). Сначала создадим в памяти пустой тег `<input>`:

```
var i = document.createElement("input");
```

По умолчанию каждое поле ввода имеет тип `"text"`; впоследствии это окажется очень важным.

Теперь присвоим атрибуту `type` пустого тега `<input>` значение, соответствующее типу, поддержку которого мы тестируем, например:

```
i.setAttribute("type", "color");
```

Если ваш браузер умеет работать с данным типом полей ввода, то в атрибуте `type` будет и далее сохраняться установленное вами значение. Если нет, то новое значение будет проигнорировано и при запросе система возвратит `"text"`:

```
return i.type !== "text";
```

Чтобы не писать самостоятельно 13 разных функций, которые бы тестировали поддержку HTML5-типов полей ввода, можно, как и в остальных случаях, воспользоваться Modernizr. В целях эффективности при тестировании всех 13 новых типов Modernizr обходится всего одним тегом `<input>`. На выходе получаем хеш `Modernizr.inputtypes` с 13 ключами (HTML5-типы полей ввода) и 13 булевыми значениями (есть поддержка — `true`, нет поддержки — `false`):

```
if (!Modernizr.inputtypes.date) {
  // встроенной поддержки <input type="date"> нет,
  // можно попробовать решить проблему с помощью Dojo или jQueryUI
}
```

Подсказывающий текст

Кроме новых типов полей ввода, HTML5 предусматривает несколько мелких дополнений к существующей системе веб-форм. Одно из них предполагает возможность назначать для всех полей ввода «подсказывающий» текст, который отображается внутри поля до тех пор, пока оно пусто и не несет фокуса. Как только пользователь щелкнет на нем или перейдет к нему с помощью табулятора, подсказывающий текст исчезнет. Если по этому описанию вам трудно представить, о чем идет речь, см. рис. 9.1.

Протестировать поддержку подсказывающего текста можно способом 2 (см. раздел «Способы тестирования браузера» этой главы). Если ваш браузер поддерживает его, то для DOM-объекта, представляющего тег <input>, будет определено свойство placeholder (даже когда в HTML-коде ему не присвоено никакое значение). Если нет, то свойство не будет определено. Проверить поможет такая функция:

```
function supports_input_placeholder() {
  var i = document.createElement('input');
  return 'placeholder' in i;
}
```

Чтобы не писать собственную функцию, прибегните к помощи Modernizr (см. раздел «Modernizr: библиотека для тестирования HTML5-функций» данной главы):

```
if (Modernizr.input.placeholder) {
  // подсказывающий текст будет виден!
} else {
  // подсказывающий текст не поддерживается,
  // вернемся к нашим сценариям
}
```

Автофокусировка в формах

На многих сайтах с помощью JavaScript реализована автоматическая фокусировка первого из полей веб-формы. Так, на главной странице Google.com форма поиска несет фокус, так что запрос можно вводить сразу, не помещая курсор в поле. Это удобно для большинства посетителей, но может мешать «продвинутым» пользователям и людям с особыми потребностями. Если на такой странице нажать **Пробел**, то прокрутка на один экран вниз не сработает, как можно ожидать; вместо этого в поле ввода появится пробел. Другой пример: если переместить фокус в другое из полей формы, пока страница еще грузится, «услужливая» система вернет курсор обратно, в начальное поле. Это сбивает с ритма, к тому же пользовательский ввод попадает не туда, куда следует.

Все эти специальные случаи трудно учесть при автофокусировке с помощью JavaScript. Еще недавно пользователей, которых раздражает «кража» фокуса веб-страницами, ничем нельзя было утешить.

Для решения проблемы в HTML5 введен атрибут autofocus, который может быть применен к любому элементу веб-формы. Этот атрибут работает в полном соответствии с названием, то есть перемещает курсор в одно из полей ввода на форме. Но это не сценарное решение, а HTML-код, значит, его поведение на всех сайтах будет одинаково. Со своей стороны, разработчики браузеров и браузерных расширений, возможно, предложат пользователям функцию отключения автофокусировки.

Протестировать поддержку автофокусировки можно способом 2 (см. раздел «Способы тестирования браузера» этой главы). Если ваш браузер поддерживает автофокусировку, то для DOM-объекта, представляющего тег <input>, будет определено свойство autofocus (даже когда в HTML-коде оно не активизируется). Если нет, то свойство не будет определено. Проверить поможет такая функция:

```
function supports_input_autofocus() {
  var i = document.createElement('input');
  return 'autofocus' in i;
}
```

Чтобы не писать собственную функцию, прибегните к помощи Modernizr (см. раздел «Modernizr: библиотека для тестирования HTML5-функций» этой главы):

```
if (Modernizr.input.autofocus) {
  // автофокусировка работает!
} else {
  // автофокусировка не работает,
  // вернемся к нашим сценариям
}
```

Микроданные

Микроданные (http://bit.ly/ckt9Rj) — это стандартизованный способ расширения семантики ваших веб-страниц. Так, например, с помощью микроданных можно указать, что та или иная фотография доступна на условиях определенной лицензии Creative Commons. Как вы увидите в главе 10, при верстке страницы с личной информацией тоже можно с успехом пользоваться микроданными. Браузеры, браузерные расширения и поисковые системы умеют преобразовывать HTML5-микроданные в формат vCard — стандарт обмена контактной информацией. Кроме всего названного, можно определять пользовательские словари микроданных.

Стандарт микроданных HTML5 включает в себя элементы разметки HTML (главным образом для поисковиков) и набор DOM-функций (главным образом для браузеров). Вашим веб-страницам не повредит разметка микроданных, ведь в каждом отдельном документе она состоит не более чем из нескольких уместно вписанных атрибутов. Поисковые системы, которые не умеют работать с микроданными, будут эти атрибуты просто игнорировать. Если же вы хотите оперировать микроданными посредством DOM, то нужно проверить, работает ли в браузере DOM API микроданных.

Протестировать поддержку API HTML5-микроданных можно способом 1 (см. раздел «Способы тестирования браузера» этой главы). Если ваш браузер поддерживает этот API, то для глобального объекта `window` будет определена функция `getItems()`. Соответственно, если поддержки микроданных в браузере нет, то функция тоже не будет определена. Проверить можно так:

```
function supports_microdata_api() {
  return !!document.getItems;
}
```

Вероятно, пока вы читали эту главу, вам успели надоесть однообразные ссылки на Modernizr. Тогда вы будете рады узнать, что Modernizr пока не умеет тестировать поддержку API микроданных, и в этом случае вам придется-таки написать проверку самостоятельно.

Для дальнейшего изучения

Спецификации и стандарты:

- тег `<canvas>` (http://bit.ly/9JHzOf);
- тег `<video>` (http://bit.ly/a3kpiq);
- типы полей ввода (http://bit.ly/akweH4);
- атрибут `<input placeholder>` (http://bit.ly/caGl8N);
- атрибут `<input autofocus>` (http://bit.ly/db1Fj4);
- локальное хранилище HTML5 (http://dev.w3.org/html5/webstorage/);
- фоновые вычисления (http://bit.ly/9jheof);
- офлайновые веб-приложения (http://bit.ly/d8ZgzX);
- API геолокации (http://www.w3.org/TR/geolocation-API/).

JavaScript-библиотеки:

- Modernizr (http://www.modernizr.com/) — библиотека для тестирования HTML5-функций;
- geo.js (http://code.google.com/p/geo-location-javascript/) — обертка для API геолокации.

Другие статьи и материалы:

- Video for Everybody! (http://camendesign.com/code/video_for_everybody);
- «Элементарное введение в кодировки видеоданных» (http://diveintomark.org/tag/give);
- «Свойства разных типов видео» (http://wiki.whatwg.org/wiki/Video_type_parameters);
- приложение к этой книге.

3 Что все это значит?

Приступим

В этой главе мы возьмем совершенно корректную веб-страницу в разметке HTML и... улучшим ее. Исходный код одних ее частей станет чуть короче, других — чуть длиннее, но главное, что вся страница придет в соответствие с идеей семантической сети. Вот увидите, какое это волшебное превращение.

Предметом наших опытов станет страница http://diveintohtml5.org/examples/blog-original.html. Вам суждено не просто изучить ее, а и буквально «сжиться» с ней, пожалуй, даже полюбить ее. Это все еще впереди. А пока заглянем в исходный код страницы.

Определение типа документа

Итак, начнем с самого начала:

```
<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

В этих строках задано так называемое *определение типа документа (ОТД)*. История возникновения ОТД долгая и довольно мрачная. При разработке браузера Internet Explorer 5 под операционную систему компьютеров Macintosh компания Microsoft столкнулась с неожиданной проблемой. В новой версии браузера было столько улучшений по части стандартов, что старые страницы отображались в нем неправильно. Вернее, отображались они правильно (в соответствии со спецификацией), но пользователь думал, что неправильно. Эти страницы были сверстаны с оглядкой на странности работы популярных браузеров того времени (2000), главным образом Netscape 4 и Internet Explorer 4. Интернет оказался не готов к той степени совершенства, которую демонстрировал IE5/Mac.

Тогда специалисты Microsoft предложили нетривиальный выход. Перед прорисовкой страницы IE5/Mac пытался найти в ее исходном тексте «тип документа», который, как правило, указывают в самых первых строках, выше открывающего тега `<html>`. Старые страницы, отображение которых надо было согласовать с причудами старых браузеров, обычно вообще не содержали ОТД. Эти страницы брау-

зер IE5/Mac прорисовывал в точности так же, как и его предшественники. Чтобы включить более полную поддержку стандартов, верстальщик должен был вписать перед тегом `<html>` то или иное ОТД.

Этот механизм быстро приобрел популярность. Вскоре во всех основных браузерах было два режима: режим совместимости (quirks mode) и стандартный (standards mode). Но затем положение снова вышло из-под контроля, как это всегда бывает в Сети. При тестировании браузера Mozilla версии 1.1 было обнаружено, что в режиме, называемом стандартным, некоторые страницы отображаются не в соответствии со спецификацией. Их прорисовка полагалась на один определенный «каприз» браузеров, и когда графический движок Mozilla после обновления перестал капризничать, внешний вид тысяч страниц был нарушен. Так возник «почти стандартный» режим (almost standards mode) — это название, как ни странно, общепринятое.

В основополагающей работе Генри Сивонена (Henri Sivonen) «ОТД и переключение режимов браузера» (http://hsivonen.iki.fi/doctype/) о разных режимах говорится следующее.

Режим совместимости

При работе в этом режиме браузер отступает от современных спецификаций, чтобы не нарушать внешний вид страниц, сверстанных по канонам конца 1990-х годов.

Стандартный режим

Работая в этом режиме, браузер пытается отобразить страницу в соответствии со спецификацией данного типа документов (и, конечно, в меру того, насколько хорошо реализованы в программе нужные для этого алгоритмы). В HTML5 стандартный режим называется режимом несовместимости (no quirks mode).

Почти стандартный режим

Браузеры Firefox, Safari, Google Chrome, Opera (начиная с версии 7.5) и IE8 также располагают «почти стандартным» режимом, в котором программы определяют высоту ячеек в таблицах не строго по спецификации CSS2, а традиционным образом. В HTML5 это называется режимом частичной совместимости (limited quirks mode).



ПРИМЕЧАНИЕ

Это очень упрощенный пересказ. Чтобы по-настоящему войти в курс дела, надо прочесть статью Сивонена целиком. Ведь на самом деле даже в IE5/Mac было предусмотрено, что некоторые (уже тогда устаревшие) ОТД не включали полную поддержку стандартов. Со временем расширялись как список известных «странностей», так и список разных ОТД, которые включают режим совместимости. В последний раз, когда я занимался подсчетами, было пять определений типа документа, переключающих браузер в «почти стандартный» режим, и еще 73, переводящих его в режим «со странностями». Вполне возможно, что какие-то из типов я не учел. О четырех (четырех!) режимах прорисовки страниц в IE8 вообще писать не буду. Как происходит переключение между ними, можно видеть на диаграмме: http://hsivonen.iki.fi/doctype/ie8-mode.png. О таких механизмах скажу только одно: их нужно уничтожать.

На чем мы остановились? Да, точно, на типе документа:

```
<!DOCTYPE html
        PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Это один из 15 типов, которые во всех современных браузерах включают стандартный режим. Такая запись корректна. Ее можно оставить, если угодно, а можно заменить определением в стиле HTML5. Оно более короткое и тоже включает стандартный режим во всех современных браузерах.

Вот это ОТД:

```
<!DOCTYPE html>
```

Как видите, всего 15 символов. Их можно не копировать, а просто напечатать — и при этом не ошибиться.

СЕКРЕТЫ РАЗМЕТКИ

Определение типа документа нужно обязательно помещать в самом начале HTML-файла. Если перед ОТД находится что-либо еще, например *одна-единственная пустая строка*, не исключено, что некоторые браузеры будут рассматривать страницу как лишенную ОТД, а значит, отображать ее в режиме совместимости. Ошибку такого рода очень трудно отследить. В HTML лишние пробелы и пустые строки обычно не играют большой роли, и глаз верстальщика не замечает их. Но в данном случае убедиться в их отсутствии очень важно!

Корневой элемент

HTML-страница представляет собой последовательность элементов (контейнеров), вложенных друг в друга. По своей структуре страница схожа с деревом. В ее исходном тексте можно найти сестринские элементы, сравнимые с двумя ветвями, которые отходят от единого ствола, а также дочерние элементы других элементов — маленькие ответвления больших ветвей (и наоборот: для элементов, непосредственно вложенных в него, контейнер является родительским элементом, а для еще более мелкого вложенного содержимого — просто предком). Элементы, не содержащие дочерних, — это «листья» дерева, а самый внешний, то есть общий предок всех элементов на странице, называется корневым. Корневой элемент HTML-страницы — это всегда `HTML` (тег `<html>`).

Корневой элемент страницы, с которой мы экспериментируем (http://diveintohtml5.org/examples/blog-original.html), выглядит следующим образом:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      lang="en"
      xml:lang="en">
```

В такой разметке опять же нет ничего ошибочного. Ее можно оставить как есть, потому что это валидный HTML5. Но стандарт HTML5 сделал ненужной часть информации, содержащейся в этом заголовке; за счет ее удаления можно сэкономить несколько байт.

Обсудим, во-первых, атрибут `xmlns` — отголосок стандарта XHTML 1.0. Он объявляет, что элементы страницы принадлежат пространству имен XHTML, описание которого находится по адресу http://www.w3.org/1999/xhtml. Но к данному пространству имен относятся все элементы HTML5; теперь уже нет нужды объявлять

это в явном виде. Вне зависимости от наличия или отсутствия атрибута xmlns страница, сверстанная на HTML5, будет одинаково показана всеми современными браузерами.

После удаления xmlns корневой элемент выглядит так:

```
<html lang="en" xml:lang="en">
```

Осталось два атрибута, lang и xml:lang. Оба указывают на язык содержания данной HTML-страницы[1]. Зачем же указывать два атрибута с одинаковым смыслом? Это снова наследие XHTML. В HTML5 действует только атрибут lang, и если вы решите оставить наряду с ним xml:lang, то следует убедиться, что в обоих назван один и тот же код языка.

> Для упрощения перехода на стандарт XHTML (и с него на более новые) в HTML-элементы иногда вводят атрибут с литералом, состоящим только из локального имени "xml:lang", без указания пространства имен и без префикса. Уместно пользоваться такой записью лишь в том случае, если xml:lang сопровождается атрибутом lang, тоже без ссылки на пространство имен и с тем же значением (сверка происходит в кодировке ASCII без внимания к регистру букв). Сам по себе атрибут с локальным именем "xml:lang" без пространства имен и без префикса никак не влияет на работу с содержимым страницы.

Готовы отказаться от ненужного? Итак, было... и не стало! Корневой элемент страницы теперь выглядит совсем просто:

```
<html lang="en">
```

Больше никаких указаний здесь не требуется.

Элемент HEAD

Первым дочерним тегом контейнера <html> обычно является <head>. Внутри <head> содержатся метаданные — это сведения о странице, а не ее собственное содержимое, «тело» (которое, естественно, заключено в контейнер <body>). Сам тег <head> при переходе к HTML5 не изменился каким-либо заметным образом, чего, однако, нельзя сказать о его содержимом. Вернемся к коду страницы-образца:

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>My Weblog</title>
  <link rel="stylesheet" type="text/css" href="style-original.css" />
  <link rel="alternate" type="application/atom+xml"
                        title="My Weblog feed"
                        href="/feed/" />
  <link rel="search" type="application/opensearchdescription+xml"
                     title="My Weblog search"
                     href="opensearch.xml" />
```

[1] Буквами en обозначается английский язык (English). Русский язык обозначался бы ru. Более подробно о кодах языков написано на сайте http://www.w3.org/International/questions/qa-choosing-language-tags.

```
  <link rel="shortcut icon" href="/favicon.ico" />
</head>
```

Начнем с рассмотрения тега <meta>.

Кодировка символов

Непрофессионал, говоря о «тексте» веб-страниц, обычно имеет в виду символы и знаки, которые можно видеть на экране компьютера. Но машина оперирует не символами и знаками, а битами и байтами. Любой текст, который вы когда-либо видели на экране, память компьютера хранит в одной из *кодировок*. Существует довольно много разных кодовых таблиц. Одни из них приспособлены под нужды конкретных языков (английский, русский, китайский...), а другие могут обслуживать множество языков. Если говорить приблизительно, то кодировка — это некоторое соответствие между множеством символов, видимых на экране, и текстовой информацией, как она хранится на жестком диске.

На деле все гораздо сложнее. Есть символы, которые имеются в разных кодировках, но соотносятся в них с разными последовательностями байт в памяти машины. Можно, таким образом, представить себе кодовую таблицу как своего рода ключ — средство расшифровки текста. Пусть некто передал вам последовательность байт и утверждает, что эта информация — текст. Тогда, чтобы расшифровать сообщение и отобразить его в виде символов на экране (подвергнуть обработке, сделать что-нибудь еще), вам надо будет узнать кодировку текста.

Каким же образом браузер определяет кодировку символов, которые веб-сервер передает ему в виде последовательности байт? Приятно, что вас интересует этот вопрос. Если вам приходилось работать с заголовками HTTP, то, возможно, вы сталкивались с записью следующего вида:

```
Content-Type: text/html; charset="utf-8"
```

Это надо понимать так: веб-сервер полагает, что он передает клиенту HTML-документ в кодировке UTF-8.

К сожалению, во всей огромной Всемирной паутине очень немногие создатели веб-страниц имеют доступ к HTTP-серверам. Так, например, на отдельных блоговых сайтах, таких как http://www.blogger.com, текстовое содержимое пишут пользователи, а за работу серверов отвечает Google. Вот почему стандарт HTML 4 предусматривал способ указать кодировку символов в самом документе. Следующая запись должна быть вам знакома:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

Это надо понимать так: создатель веб-страницы полагает, что опубликованный им документ содержит текст в кодировке UTF-8.

В HTML5 действуют оба описанных способа. Предпочтительно пользоваться заголовком HTTP, который (если он есть) переопределяет содержимое тега <meta>. Но вписывать заголовки HTTP разрешено не всем, поэтому свою роль выполняет и <meta>. Обращаться с ним в HTML5 стало чуть легче; теперь можно написать так:

```
<meta charset="utf-8" />
```

Эта сокращенная запись работает во всех браузерах. Вот лучшее объяснение ее целесообразности из всех, что мне удалось найти (http://lists.w3.org/Archives/Public/public-html/2007Jul/0550.html):

> Комбинация <meta charset=""> удобна потому, что в клиентских приложениях она фактически уже реализована, ведь кавычки при верстке часто опускают, например:
>
> `<META HTTP-EQUIV=Content-Type CONTENT=text/html; charset=ISO-8859-1>`

Есть несколько тестов на распознавание `<meta charset>` (http://simon.html5.org/test/html/parsing/encoding), к которым вы можете прибегнуть, если хотите проверить, что браузеры уже поддерживают такую форму записи.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Я никогда не пользуюсь странными буквами иностранных алфавитов. Неужели мне все-таки надо каждый раз объявлять кодировку текста?

Ответ: Да, конечно, каждый раз, на каждой сверстанной вами HTML-странице! Если не указать кодировку, это может создать уязвимость в защите сайта (http://code.google.com/p/doctype/wiki/ArticleUtf7).

Подведем итоги. Кодировками пользоваться непросто, и работать с ними не стало проще за те два или три десятка лет, в течение которых горе-авторы, виртуозы копирования и вставки, злоупотребляли услугами плохих программ — текстовых процессоров. Но, чтобы избежать нежелательных последствий, обязательно указывайте кодировку символов в коде каждого HTML-документа. Это можно сделать с помощью HTTP-заголовка `Content-Type`, или в объявлении `<meta http-equiv>`, или более компактно в объявлении `<meta charset>`. Только, пожалуйста, не забудьте. Пользователи Интернета будут вам признательны.

Ссылочные отношения

Обычная ссылка (`<a href>`) просто указывает на другую страницу. А вот ссылочные отношения — способ вместе с тем пояснить, почему надо указать именно на эту страницу. Итак, «я ссылаюсь на данную страницу, потому что...»:

- ...это таблица стилей, содержащая CSS-правила, которые браузер должен применить к данной странице;
- ...это RSS-канал с тем же содержимым, что и страница, но в естественном для подписки на новости формате;
- ...это перевод данной страницы на другой язык;
- ...это то же самое содержимое, но в формате PDF;
- ...это следующая глава книги, опубликованной в Сети, и т. д.

В HTML5 ссылочные отношения подразделяются на две категории (http://bit.ly/d2cbiR):

> С помощью элемента LINK можно создавать два типа ссылок. Ссылки на внешние ресурсы отсылают к файлам, встраиваемым в текущий документ, а гипертекстовые ссылки ведут к другим документам. [...]
> Каким образом ведет себя ссылка на внешний ресурс, зависит от типа объявленного ею отношения.

Из приведенных выше примеров только первый (`rel="stylesheet"`) представляет собой ссылку на внешний ресурс, а все остальные — гипертекстовые ссылки. Неважно, пожелает ли пользователь перейти по такой ссылке: на отображении текущей страницы это никак не скажется. Ссылочные отношения чаще всего задаются тегами `<link>` внутри контейнера `<head>`. Позволено определять атрибут `rel` в теге `<a>`, но это довольно редкий прием верстки. В HTML5 разрешено также указывать отношения в тегах `<area>`, что еще более редко встречается на практике, так как в HTML4 такая возможность отсутствовала. Полная таблица отношений (http://bit.ly/a3nsqi) подскажет вам, может ли быть использовано то или иное значение атрибута `rel`.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Можно ли создавать свои собственные ссылочные отношения?

Ответ: Вводить новые ссылочные отношения предлагают многие разработчики; приток идей в этой области поистине неиссякаем. Чтобы не допустить путаницы, рабочая группа WHAT ведет реестр предложенных значений rel (http://wiki.whatwg.org/wiki/RelExtensions) и определяет процедуру утверждения (http://bit.ly/da3pse).

rel = stylesheet

Рассмотрим первое ссылочное отношение в исходном коде нашей страницы:

```
<link rel="stylesheet" href="style-original.css" type="text/css" />
```

Это, без преувеличения, наиболее популярное в мире ссылочное отношение. Тег `<link rel="stylesheet">` указывает на подключаемый файл с таблицей стилей CSS. В духе HTML5 было бы внести мелкую упрощающую правку: убрать атрибут `type`, ведь в Интернете для описания стилей страниц используется только язык CSS. Останется:

```
<link rel="stylesheet" href="style-original.css" />
```

Во всех браузерах эта запись работает (если вдруг кто-нибудь изобретет новый язык описания стилей, можно будет снова вписать атрибут `type`).

rel = alternate

Вернемся к странице-образцу:

```
<link rel="alternate"
      type="application/atom+xml"
      title="My Weblog feed"
      href="/feed/" />
```

Это ссылочное отношение тоже довольно широко распространено. Тег `<link rel="alternate">` с атрибутом `type`, задающим формат новостного канала RSS или Atom, обеспечивает так называемое автоматическое распознавание каналов. Благодаря этому указанию система агрегации лент, например Google Reader, узнает, что на сайте есть новостной канал, передающий самые свежие записи. В большинстве браузеров browser автораспознавание каналов тоже поддерживается: если найдена новостная лента, то рядом с URL-адресом страницы будет отображаться специальный значок (здесь, в отличие от `rel="stylesheet"`, атрибут `type` важен. Не удаляйте его!).

В HTML 4 отношение `rel="alternate"` использовалось самыми разнообразными и неожиданными способами. В HTML5 его смысл уточнен и расширен для более точного описания существующего веб-содержимого. Как было только что показано, отношение `rel="alternate"` в сочетании с `type="application/atom+xml"` указывает на новостной канал Atom, передающий обновления рассматриваемой страницы. Сочетая `rel="alternate"` с другими значениями атрибута `type`, можно ссылаться на содержимое такого же рода в других форматах, например PDF.

В HTML5, кроме того, решена одна старая проблема: язык страницы-перевода. HTML 4 рекомендует пользоваться в данном случае ссылочным отношением `rel="alternate"` и атрибутом `lang`, но это неверно. Список замеченных неточностей (Errata) стандарта HTML 4 упоминает, помимо нескольких мелких недосмотров, о четырех ошибках в спецификации. Одна из них касается способа указания языка документа, на который ведет ссылка с `rel="alternate"`. Список неточностей, как и современный стандарт HTML5, свидетельствует о том, что нужно пользоваться атрибутом `hreflang` (к сожалению, этот список так и не был введен в состав спецификации HTML 4, потому что после окончания полномочий соответствующей рабочей группы W3C никто из ее членов не занимался стандартом HTML).

Другие ссылочные отношения HTML5

Рассмотрим и другие ссылочные отношения.

- `rel="archives"` (http://bit.ly/clzlyG) — «показывает, что документ, к которому ведет ссылка, представляет собой свод записей, документов или иных материалов, имеющих исторический интерес. Главная страница блога может ссылаться на архив записей с помощью отношения `rel="archives"`».
- `rel="author"` — используется для ссылки на информацию об авторе страницы. Это может быть, например, адрес электронной почты. Ссылка может просто вести на форму «Связаться» или страницу «Об авторе».
- `rel="external"` (http://bit.ly/dBVO09) — «показывает, что документ, к которому ведет ссылка, находится на стороннем сайте». Популярность этому отношению придала, как мне кажется, система WordPress, в которой таким способом маркировались внешние ссылки, оставленные в комментариях к записям.
- `rel="start"`, `rel="prev"` и `rel="next"` (http://www.w3.org/TR/html401/types.html#type-links) — предназначены для того, чтобы описывать отношения между страницами, которые являются частью единой последовательности, например главами книги или блоговыми записями. Разработчики в большинстве случаев умеют правильно пользоваться только `rel="next"`. Вместо `rel="prev"` писали `rel="previous"`, вместо

rel="start" — rel="begin" и rel="first", а взамен rel="last" — rel="end". Кроме того, для ссылки на родительскую страницу веб-программисты придумали обозначение rel="up".

В стандарт HTML5 введено отношение rel="first" — самый распространенный из всех способов обозначить первую страницу последовательности (синоним rel="start" оставлен только в целях обратной совместимости и не отвечает новой спецификации). Как и в HTML4, в пришедшем ему на смену стандарте сохранились rel="prev" и rel="next" (а также, для обратной совместимости, введено отношение rel="previous"). Добавлены rel="last" (заключительная страница последовательности) и rel="up".

Чтобы понять суть rel="up", посмотрите на дублирующую навигационную панель или хотя бы представьте себе ее. «Хлебные крошки» начинаются с главной страницы сайта и заканчиваются той страницей, на которой вы сейчас находитесь. В этой ситуации rel="up" будет указывать на предпоследнюю страницу ряда.

- rel="icon" (http://bit.ly/diAJUP) — это второе по популярности после rel="stylesheet" ссылочное отношение (http://code.google.com/webstats/2005-12/linkrels.html). Обычно оно указывает на значок, например, так:

```
<link rel="shortcut icon" href="/favicon.ico">
```

  Связывать таким образом значок со страницей позволяют все основные браузеры. Маленькая картинка favicon.ico отобразится или в адресной строке браузера рядом с URL, или в заголовке вкладки, или и там и там. В HTML5 есть важное нововведение: теперь со ссылочным отношением rel="icon" можно использовать атрибут sizes, определяющий размер изображения, на которое ссылается страница (http://bit.ly/diAJUP).

- rel="license" (http://bit.ly/9n9Xfv) — было придумано сторонниками микроформатов. Отношение «показывает, что документ, к которому ведет ссылка, представляет собой лицензионное соглашение, на условиях которого доступна данная страница».

- rel="nofollow" (http://bit.ly/cGjSPi) — «показывает, что ссылка указана не лицом, которое создало или опубликовало страницу, или что постановка ссылки обусловлена в первую очередь коммерческими связями между владельцами этих двух страниц». Отношение rel="nofollow" придумали специалисты Google; в сообществе разработчиков микроформатов это ссылочное отношение было стандартизовано. Полагали, что если алгоритм PageRank не будет учитывать ссылки с "nofollow", то спамеры потеряют интерес к комментированию блогов. Этого не произошло, но rel="nofollow" сохранило свою роль. Многие блог-сервисы по умолчанию добавляют rel="nofollow" к ссылкам, которые публикуются комментаторами записей.

- rel="noreferrer" (http://bit.ly/cQMSJg) — «показывает, что никакая информация о ссылающейся странице не должна быть передана при переходе по ссылке». Эта функция пока не работает ни в одном популярном браузере, но ее поддержка была недавно добавлена в «ночные сборки» движка WebKit, так что следует ожидать, что когда-нибудь rel="noreferrer" будут поддерживать

Safari, Google Chrome и другие браузеры на основе WebKit. Работает ли rel="noreferrer" в вашем браузере, можно проверить на http://wearehugh.com/public/2009/04/rel-noreferrer.html.

- rel="pingback" (http://bit.ly/cIAGXB) — указывает адрес пингбэк-сервера. Как поясняет спецификация (http://hixie.ch/specs/pingback/pingback-1.0), «система пингбэка — это способ автоматически уведомлять владельца блога о ссылках на его страницы с других сайтов. [...] Это возможность ставить обратные ссылки, по цепочке которых можно проходить не только вперед, как обычно, но и назад». На блоговых сервисах, в частности WordPress, механизм пингбэка реализован так, чтобы уведомлять авторов о ссылках на их посты в постах других авторов.
- rel="prefetch" (http://bit.ly/9o0nMS) — «показывает, что могут быть удобны предварительная загрузка и кэширование данного ресурса, так как весьма вероятно, что он будет востребован пользователем». В том случае, когда самый популярный результат поиска по какому-либо запросу превосходит другие по частоте обращений во много раз, поисковая система может выдать ссылку такого вида: <link rel="prefetch" href="<emphasis>URL-адрес самого популярного результата</emphasis>">. Так, если в Firefox поискать через Google по запросу CNN, в коде страницы результатов обнаружится слово prefetch. В настоящее время только Mozilla Firefox поддерживает rel="prefetch".
- rel="search" (http://bit.ly/aApkaP) — «показывает, что документ, к которому ведет ссылка, предоставляет интерфейс поиска по данному документу и родственным ресурсам». Если вы хотите применять rel="search" с пользой для себя, поставьте ссылку на документ OpenSearch, описывающий правила, в соответствии с которыми браузер может построить URL-адрес поискового запроса по сайту на какое-либо слово. Технологию OpenSearch и ссылочные отношения rel="search", которые указывают на OpenSearch-документацию, поддерживают Internet Explorer (начиная с версии 7) и Mozilla Firefox (с версии 2).
- rel="sidebar" (http://bit.ly/azTA9D) — «показывает, что документ, к которому ведет ссылка, должен отображаться (если пользователь перейдет по ссылке) не в основном, а во второстепенном контексте». Для браузеров Opera и Mozilla Firefox это значит буквально следующее: когда пользователь щелкнет на ссылке, система предложит ему создать закладку для страницы, которая, будучи открыта через меню Закладки, отобразится на боковой панели браузера (sidebar, в английской терминологии Opera — panel). В Internet Explorer, Safari и Chrome ссылки с rel="sidebar" функционируют как обычные ссылки. Протестировать работу rel="sidebar" в своем браузере вы можете на странице http://wearehugh.com/public/2009/04/rel-sidebar.html.
- rel="tag" (http://bit.ly/9bYlfa) — «показывает, что документ, к которому ведет ссылка, представляет некоторую категорию и текущий документ относится к этой категории». Разметка с помощью категорий (тегов, ключевых слов) в атрибуте rel была придумана программистами Technorati, чтобы удобнее было группировать блоговые записи по темам. Поэтому в старых публикациях можно встретить такой термин, как «теги Technorati» (да, все именно так: коммерческая компания сумела убедить весь мир в пользе метаданных, которые упрощают работу самой

этой компании!). В сообществе разработчиков микроформатов был впоследствии стандартизован упрощенный синтаксис `rel="tag"`. На большинстве таких блоговых сервисов, где разрешено связывать отдельные записи с категориями или ключевыми словами, используется именно ссылочное отношение `rel="tag"`. Браузеры не делают с такими ссылками ничего особенного, а вот поисковики судят по ним о тематике страниц.

Новые семантические элементы HTML5

Приведение разметки к стандарту HTML5 не только укорачивает запись. Это шанс воспользоваться набором новых семантических элементов. Рассмотрим теги, определенные в спецификации HTML5.

- <section> — обобщенно понимаемый раздел документа или приложения. В данном контексте раздел — это порция содержимого, сгруппированная по тематическому признаку и, как правило, оснащенная заголовком. Примеры разделов — главы в книгах, вкладки в диалоговых окнах, пронумерованные части научных публикаций. Главная страница сайта может содержать несколько разделов: вводные замечания, новости, контактная информация и пр.
- <nav> — раздел, содержащий ссылки на другие страницы или на части данной страницы, то есть навигационный раздел. Тегом <nav> следует оформлять не все группы ссылок на странице; его назначению отвечают только большие навигационные блоки. В частности, нижний колонтитул обычно содержит несколько ссылок на разные страницы сайта: соглашение об использовании, главная страница, информация об авторах... Для верстки в этом случае подойдет тег <footer>, а надобности в <nav> нет.
- <article> — фрагмент страницы, документа, изображения или целого сайта, который самостоятельно значим, может независимо распространяться или повторно использоваться (например, в агрегаторах лент). Это может быть запись на форуме, в блоге, статья в журнале или газете, пользовательский комментарий, интерактивный виджет или гаджет, а также любая другая независимая единица контента.
- <aside> — раздел страницы, содержимое которого лишь косвенно связано с темой всего прочего содержимого документа и может рассматриваться отдельно от него. В полиграфии такие разделы зачастую оформляются как врезки. Тег <aside> можно использовать для создания «типографских» эффектов врезки, выделенной цитаты, для рекламных блоков (теглайнов), групп тегов <nav> и для прочего неосновного содержимого страницы.
- <hgroup> — заголовок раздела. Контейнер <hgroup> используется для группировки тегов <h1>…<h6>, которые представляют собой заголовки разных уровней: основной, подзаголовок, рекламный слоган, второе название.
- <header> — верхний колонтитул. Это группа вспомогательных материалов вводного или навигационного характера. Считается желательным, но не обязательным, чтобы тегом <header> был оформлен заголовок раздела — группа тегов

<h1>…<h6> или тег <hgroup>. Кроме того, <header> иногда содержит оглавление раздела, форму поиска и логотип (-ы).

- <footer> — нижний колонтитул родительского элемента: раздела или всей страницы. Обычно содержит информацию о разделе и его создателе, ссылки на родственные документы, замечания об авторских правах и т. п. Нижний колонтитул не всегда, но обыкновенно находится в конце раздела. Он может и сам содержать несколько разделов, в качестве которых, как правило, выступают приложения, указатели, выходные данные, многословные лицензионные соглашения и другое содержимое подобного рода.
- <time> — может содержать указание времени по 24-часовой шкале или точной даты по григорианскому календарю, в запись которой можно также включить время и часовой пояс.
- <mark> — подсвечивает на странице какой-либо важный текст, к которому пользователь, возможно, еще пожелает вернуться.

Понимаю, что вам не терпится приступить к использованию новых элементов, иначе вы бы уже давно бросили читать эту главу. Однако отвлечемся еще ненадолго.

Большое отступление о том, как браузеры обрабатывают незнакомые элементы

Каждый браузер «знает», поддержка каких HTML-элементов в нем совершенно точно реализована. Например, браузер Mozilla Firefox хранит список таких элементов в файле nsElementTable.cpp (http://mxr.mozilla.org/seamonkey/source/parser/htmlparser/src/nsElementTable.cpp). Элементы, не входящие в этот список, расцениваются программой как незнакомые. Для обработки незнакомых элементов надо ответить на два вопроса.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Какой стиль следует применить к элементу?

Так, например, контейнер <p> имеет отступы сверху и снизу, <blockquote> — поле слева, а заголовок <h1> отображается более крупным шрифтом, чем основной текст.

Каково место элемента в иерархии DOM?

В файле nsElementTable.cpp браузера Mozilla Firefox содержится информация о том, какие типы элементов может принимать как дочерние каждый из знакомых элементов. Запись вида <p><p> означает с точки зрения браузера, что перед началом второго абзаца первый заканчивается: это элементы-сестры (а не родительский и дочерний). Но если написать <p><span>, то <span> не закроет абзац, потому что (как знает Firefox) <p> — это блочный тег, а <span> — строковый. Значит, в DOM-иерархию <span> будет помещен как дочерний относительно <p>.

Разные браузеры по-разному отвечают на эти вопросы (непривычного человека такое откровение может обескуражить). Из числа популярных браузеров больше всего трудностей в этом деле испытывает Internet Explorer.

Ответить на первый вопрос вроде бы довольно просто. Незнакомым элементам не следует приписывать какой-либо особый стиль, а надо просто позволить

унаследовать стили, описанные для них в явном виде в CSS-таблице при верстке. Однако Internet Explorer версий 6, 7 и 8 не применяет стили к незнакомым элементам. Так, если в коде страницы написано:

```
<style type="text/css">
  article { display: block; border: 1px solid red }
</style>
...
<article>
  <h1>Добро пожаловать в Ад¹</h1>
  <p>Вы здесь <span>первый день</span>.</p>
</article>
```

то Internet Explorer (до IE8 включительно) не обведет содержимое элемента ARTICLE рамкой красного цвета. Когда писались эти строки, Internet Explorer 9 еще пребывал в стадии бета-тестирования, но Microsoft заявляет, что в 9-й версии таких проблем не будет (это подтверждают и разработчики).

Вторая проблема — построение DOM-иерархии. С этим Internet Explorer тоже справляется хуже других популярных браузеров. Если IE не распознает элемент как знакомый, он вставит его в DOM как пустой узел без потомков и все элементы, которые, по мнению веб-программиста, должны выступать как дочерние по отношению к нему, окажутся его сестрами.

Покажем различие с помощью несложных схем. Вот DOM-дерево, построенное в соответствии с требованиями HTML5:

```
article
|
+—h1 (дочерний узел article)
|  |
|  +—текстовый узел "Добро пожаловать в Ад"
|
+—p (дочерний узел article, сестра h1)
  |
  +—текстовый узел "Вы здесь "
  |
  +—span
  |   |
  |   +—текстовый узел "первый день"
  |
  +—текстовый узел "."
```

А вот DOM-дерево, которое строит Internet Explorer:

```
article (нет потомков)
h1 (сестра article)
|
+—текстовый узел "Добро пожаловать в Ад"
p (сестра h1)
```

[1] В оригинале упомянута вымышленная компания Initech из комедии «Офисное пространство», малоизвестной у нас. — *Примеч. перев.*

```
|
+—текстовый узел "Вы здесь "
|
+—span
| |
| +—текстовый узел "первый день"
|
+—текстовый узел "."
```

Существует удивительный способ устранить эту проблему. Если с помощью JavaScript создать пустой тег <article>, то, прежде чем этот тег в действительности будет задействован на странице, Internet Explorer волшебным образом обнаружит <article> и применит к нему нужный стиль. В DOM-иерархию этот пустой тег не попадает, однако достаточно одной вставки (на страницу), чтобы IE «научился» правильно оформлять все незнакомые элементы такого типа на странице. Образец:

```
<html>
  <head>
    <style>
      article { display: block; border: 1px solid red }
    </style>
    <script>document.createElement("article");</script>
  </head>
  <body>
    <article>
      <h1>Добро пожаловать в Ад</h1>
      <p>Вы здесь <span>первый день</span>.</p>
    </article>
  </body>
</html>
```

Такая запись работает корректно во всех версиях Internet Explorer, вплоть до IE6.

Более того, можно вставить на страницу по одному разу все элементы HTML5. Они не попадают в DOM, и пользователь их не видит, но за счет такой вставки веб-мастер может прибегать к новым возможностям, не беспокоясь об отображении страниц в браузерах без поддержки HTML5. Эту задачу решил Реми Шарп (Remy Sharp) (http://remysharp.com/2009/01/07/html5-enabling-script). Его сценарий имеет несколько версий, код которых в упрощенном виде выглядит одинаково:

```
<!--[if lt IE 9]>
<script>
  var e = ("abbr,article,aside,audio,canvas,datalist,details," +
    "figure,footer,header,hgroup,mark,menu,meter,nav,output," +
    "progress,section,time,video").split(',');
  for (var i = 0; i < e.length; i++) {
    document.createElement(e[i]);
  }
</script>
<![endif]-->
```

Строки `<!--[if lt IE 9]>` и `<![endif]-->` открывают и закрывают условный комментарий соответственно. Internet Explorer видит в них утверждение: «Если текущий браузер — Internet Explorer вплоть до 8-й версии, то исполнить следующий код». Все остальные браузеры обращаются с условным комментарием, как и с любым другим HTML-комментарием. В результате Internet Explorer вплоть до 8-й версии запускает сценарий, а любой другой браузер игнорирует его, что ускоряет загрузку страницы.

Сам по себе код на JavaScript, приведенный выше, довольно прямолинеен. В переменную `e` заносится массив строк вида `"abbr"`, `"article"` и т. д. Цикл перебирает эти строки (названия HTML5-элементов) и командой `document.createElement()` создает соответствующие пустые элементы. Поскольку возвращаемые значения не присваиваются никакой переменной, элементы не попадают в DOM. Однако для Internet Explorer этого достаточно: когда далее на странице будут встречаться непустые элементы тех же типов, они будут правильно отображаться.

Обратите внимание: «далее на странице». Сценарий должен располагаться в начале документа и желательно внутри контейнера `<head>`, а не в его конце. Таким образом, Internet Explorer будет сначала исполнять сценарий и только *потом* разбирать теги и атрибуты. Если убрать сценарий далеко вниз, то будет уже поздно: Internet Explorer неверно интерпретирует разметку страницы и ошибочно построит DOM-иерархию. Повторный разбор кода страницы выполняться не будет.

«Облегченную» версию своего сценария Реми Шарп поместил в репозиторий Google Code (http://code.google.com/p/html5shiv/). Сценарий распространяется под лицензией MIT с открытым исходным кодом, так что вы можете пользоваться им в любом своем проекте. Можно даже ссылаться непосредственно на версию, хранимую Google:

```
<head>
  <meta charset="utf-8" />
  <title>Мой блог</title>
  <!--[if lt IE 9]>
  <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
  <![endif]-->
</head>
```

Теперь мы готовы приступить к использованию новых семантических элементов HTML5.

Верхние колонтитулы

Вернемся к странице, с которой мы экспериментируем, и обратим внимание на верхний колонтитул[1]:

```
<div id="header">
  <h1>Мой блог</h1>
```

[1] Основной английский текст веб-страниц дается здесь и далее в русском переводе. — *Примеч. перев.*

```
  <p class="tagline">Только сложные пути ведут к простым решениям.</p>
</div>
...
<div class="entry">
  <h2>День в дороге</h2>
</div>
...
<div class="entry">
  <h2>Я еду в Прагу!</h2>
</div>
```

Такая верстка вполне корректна. Если угодно, ее можно оставить как есть, потому что это валидный HTML5. Однако в HTML5, как мы знаем, есть особые семантические элементы для колонтитулов и разделов.

Первым делом избавимся от <div id="header">. Это популярная форма записи, но она ничего не значит. У тега <div> нет собственной семантики, как и у атрибута id (это, конечно, взгляд с точки зрения клиентских программ, которые не умеют судить о содержании информационного блока по значению атрибута id). Можно было бы назвать тот же контейнер <div id="preved"> — и ничего бы не изменилось.

В HTML5 для оформления верхних колонтитулов есть тег <header>. Спецификация приводит много примеров использования <header> в прикладных задачах верстки. Код нашей страницы после внесенной правки будет выглядеть так:

```
<header>
  <h1>Мой блог</h1>
  <p class="tagline">Только сложные пути ведут к простым решениям.</p>
  ...
</header>
```

Стало лучше, правда? Теперь из кода ясно, что перед нами верхний колонтитул. Пора взяться за подзаголовок-теглайн, который здесь сверстан обычным, но не стандартизованным (как это теперь стало возможно) способом. Оформить теглайн — непростая задача: он должен вести себя как подзаголовок, привязанный к основному заглавию, то есть подзаголовок без следующего за ним раздела.

Заголовочные теги, такие как <h1> и <h2>, определяют структуру страницы. Взятые в совокупности, они формируют краткое содержание документа, в соответствии с которым может быть организована навигация по странице. Программы экранного доступа, опираясь на краткое содержание, помогают слепым пользователям ориентироваться на страницах. Есть веб-сервис (http://gsnedders.html5.org/outliner/) и браузерное расширение (http://chrispederick.com/work/web-developer/), которые умеют графически представлять краткое содержание страниц.

В HTML 4 построить краткое содержание страницы можно было только на основе тегов <h1>…<h6>. Схема страницы-образца выглядит таким образом:

```
Мой блог (h1)
|
+—День в дороге (h2)
|
+—Я еду в Прагу! (h2)
```

Здесь все в порядке, но теглайну «Только сложные пути ведут к простым решениям» места не нашлось. Если бы мы поместили его в контейнер <h2>, в кратком содержании документа появился бы пустой узел:

```
Мой блог (h1)
|
+—Только сложные пути ведут к простым решениям. (h2)
|
+—День в дороге (h2)
|
+—Я еду в Прагу! (h2)
```

На самом же деле структура документа не такая. Теглайн не является заголовком одной из записей блога: это подзаголовок, обособленно помещенный на страницу. А если спрятать теглайн в <h2>, а заголовок каждой записи — в <h3>? Нет, так будет только хуже:

```
Мой блог (h1)
|
+—Только сложные пути ведут к простым решениям. (h2)
   |
   +—День в дороге (h3)
   |
   +—Я еду в Прагу! (h3)
```

В кратком содержании документа по-прежнему есть пустой узел, который к тому же присвоил дочерние узлы, по праву принадлежащие корню иерархии. В этом вся незадача: HTML 4 не позволяет оформить теглайн так, чтобы он, будучи одним из подзаголовков, тем не менее не попадал в краткое содержание страницы. Как бы мы ни старались перехитрить компьютер, слова «Только сложные пути ведут к простым решениям» неминуемо осядут в одном из узлов структуры. Вот почему семантически бессмысленную верстку вида <p class="tagline"> мы сочли меньшим злом.

Решение, которое предлагает HTML5, заключается в применении тега <hgroup>. Он функционирует в качестве обертки для двух или нескольких связанных заголовков. Называя их связанными, мы имеем в виду, что в кратком содержании документа они образуют единый узел. При следующем коде страницы:

```
<header>
  <hgroup>
    <h1>Мой блог</h1>
    <h2>Только сложные пути ведут к простым решениям.</h2>
  </hgroup>
  ...
</header>
...
<div class="entry">
  <h2>День в дороге</h2>
</div>
...
<div class="entry">
```

```
  <h2>Я еду в Прагу!</h2>
</div>
```

краткое содержание будет выглядеть таким образом:

```
Мой блог (h1 в составе hgroup)
|
+—День в дороге (h2)
|
+—Я еду в Прагу! (h2)
```

Чтобы убедиться, что вы правильно пользуетесь заголовками, тестируйте создаваемые вами страницы на сайте HTML5 Outliner.

Рубрикация

Разберемся, как можно преобразовать дальнейший код:

```
<div class="entry">
  <p class="post-date">22 октября 2009 г.</p>
  <h2>
    <a href="#"
       rel="bookmark"
       title="ссылка на эту запись">
       День в дороге
    </a>
  </h2>
  ...
</div>
```

Это тоже валидный HTML5. Однако в HTML5 есть тег <article>, использование которого (в общем случае) при разбивке страницы на статьи или записи более оправданно:

```
<article>
  <p class="post-date">22 октября 2009 г.</p>
  <h2>
    <a href="#"
       rel="bookmark"
       title="ссылка на эту запись">
       День в дороге
    </a>
  </h2>
  ...
</article>
```

Однако все не так просто, и понадобится внести еще одно изменение, которое я сначала покажу, а потом объясню:

```
<article>
  <header>
    <p class="post-date">22 октября 2009 г.</p>
```

```
    <h1>
      <a href="#"
         rel="bookmark"
         title="ссылка на эту запись">
         День в дороге
      </a>
    </h1>
  </header>
  ...
</article>
```

Ясно? Тег <h2> был заменен тегом <h1> в обертке <header>. Как работает тег <header>, вы уже видели. В данном случае его функция — заключать в себя все элементы верхнего колонтитула записи (дату публикации и название). Но... как быть с тем, что в документе должен быть только один заголовок <h1>? Не пострадает ли от нашей правки краткое содержание документа? Нет. Чтобы понять почему, вернемся на шаг назад.

В HTML 4 краткое содержание строилось с опорой *только* на теги <h1>...<h6>. Если веб-мастер хотел видеть в смысловой схеме документа один корневой узел, он должен был ограничиться лишь одним заголовком <h1>. Но спецификация HTML5 задает новый алгоритм построения краткого содержания с учетом современных семантических тегов. Согласно этому алгоритму, тег <article> создает новый раздел, то есть очередной дочерний узел корня краткого содержания. Между тем в HTML5 внутри каждого раздела может иметься свой собственный тег <h1>.

Итак, произошла большая и, как мы сейчас увидим, благотворная перемена. Действительно, многие веб-страницы сверстаны по образцам: часть содержимого берется из одного источника и вставляется куда-либо на страницу, другой фрагмент (из другого источника) помещается еще куда-то. Этот подход поощряют многие руководства по гипертекстовой разметке: «Вот кусок HTML-кода, скопируйте его и вставьте на страницу». Если копируется небольшая порция кода, то проблем обычно не возникает; но как быть с копированием и вставкой целых разделов? Тогда в руководствах писали, к примеру, так: «Вот кусок HTML-кода. Скопируйте его, вставьте в текстовый редактор и отредактируйте теги-заголовки, чтобы они согласовывались по вложенности с соответствующими заголовками страницы, на которую вы затем будете вставлять данный код».

В HTML 4 не было *обобщенного* заголовочного элемента, а было шесть заголовков от <h1> до <h6>: числа в названиях тегов строго определяли порядок вложения. Контролировать такую иерархию, мягко говоря, неудобно, особенно на странице, которая не «написана» (в собственном смысле), а «собрана» из фрагментов. Эта проблема решена в HTML5 благодаря новым элементам, вводящим разделы, и новым правилам использования существующих тегов-заголовков. Если вы не боитесь применять семантические элементы верстки, советую писать так:

```
<article>
  <header>
    <h1>Моя запись в блоге</h1>
  </header>
  <p>Lorem ipsum… (и далее по тексту классического текста-"рыбы")</p>
</article>
```

Этот фрагмент кода можно копировать и вставлять *в любое место страницы*, ничего в нем не меняя. Имеющийся здесь тег <h1> не предоставляет проблем, потому что целое заключено в контейнер <article>. Тег <article> объявляет самостоятельный узел в кратком содержании документа, тег <h1> задает имя этого узла; уровень вложенности всех прочих элементов страницы, отвечающих за дробление на разделы, остается таким, каким должен быть.

СЕКРЕТЫ РАЗМЕТКИ

В действительности все чуть сложнее, чем описано здесь. Новая система тегов «явной» рубрикации, таких как <h1> в обертке <article>, может весьма неожиданным образом взаимодействовать со старой системой «скрытой» рубрикации (собственно заголовками <h1>...<h6>). Использование одной из этих двух методик верстки, а не обеих сразу существенно облегчит ваш труд. Если зачем-либо надо сделать на одной странице «явные» и «скрытые» разделы, не поленитесь проверить с помощью HTML5 Outliner, правильно ли выглядит краткое содержание вашего документа.

Дата и время

Захватывающе, правда? Не так захватывающе, конечно, как нагишом съезжать с Эвереста на лыжах, но ровно настолько, насколько вообще может быть захватывающей семантическая верстка. Вернемся к странице-образцу и обратим внимание на следующие строки:

```
<div class="entry">
  <p class="post-date">22 октября 2009 г.</p>
  <h2>День в дороге</h2>
</div>
```

Все то же самое, правда? Указывать дату публикации статьи или записи в блоге — обычная практика, но для этого до недавних пор не существовало семантического тега и верстальщику приходилось прибегать к обобщенной форме записи с нестандартным значением атрибута class. В HTML5 такая запись по-прежнему работает, в ней не обязательно что-либо менять. Но специально для таких случаев стандарт HTML5 предусматривает тег <time>:

```
<time datetime="2009-10-22" pubdate>22 октября 2009 г.</time>
```

В теге <time> три составные части:

- метка времени в машиночитаемом формате;
- текстовое содержимое, которое отображается для читателя-человека;
- необязательный флажок pubdate.

В нашем примере атрибут datetime содержит информацию только о дате, без указания времени. Формат даты — ГГГГ-ММ-ДД:

```
<time datetime="2009-10-22" pubdate>22 октября 2009 г.</time>
```

Чтобы добавить информацию о времени, допишите после даты букву T, время в 24-часовом формате (ЧЧ:ММ:СС) и далее, через дефис, разницу поясного времени с Гринвичем:

```
<time datetime="2009-10-22T13:59:47-04:00" pubdate>
  22 октября 2009 г. 13:59 EDT[1]
</time>
```

Формат представления даты и времени очень гибок. Спецификация HTML5 содержит множество образцов корректно оформленных значений атрибута datetime.

Вы, наверное, заметили, что текст, заключенный между <time> и </time>, в последнем примере поменялся вместе с машиночитаемой меткой времени. Стандарт не требует этого: если вы проставили метку времени (значение атрибута datetime), то текстовое содержимое тега <time> может быть каким угодно. В HTML5 валиден такой код:

```
<time datetime="2009-10-22">Дело было в прошлый четверг</time>
```

И даже такой:

```
<time datetime="2009-10-22"></time>
```

Осталось рассмотреть атрибут pubdate. Он булевозначен, то есть играет роль его наличие или отсутствие. Верна такая запись:

```
<time datetime="2009-10-22" pubdate>22 октября 2009 г.</time>
```

Если вам не по душе атрибуты без значений, можно (равносильно) написать так:

```
<time datetime="2009-10-22" pubdate="pubdate">22 октября 2009 г.</time>
```

Атрибут pubdate значит одно из двух. Если тег <time> заключен внутрь контейнера <article>, то при указанном pubdate метка времени — это дата публикации статьи или записи. Если же <time> не отнесен ни к одному из разделов <article>, то при заданном pubdate метка времени указывает на дату публикации всего документа.

Вот как использование новых возможностей HTML5 изменит верстку нашей условной записи в блоге:

```
<article>
  <header>
    <time datetime="2009-10-22" pubdate>
      22 октября 2009 г.
    </time>
    <h1>
      <a href="#"
         rel="bookmark"
         title="ссылка на эту запись">
         День в дороге
      </a>
    </h1>
  </header>
  <p>Lorem ipsum… (и далее по тексту классического текста-"рыбы")</p>
</article>
```

[1] EDT (Eastern Daylight Time) — единое время Восточного побережья США. — *Примеч. перев.*

Навигация

Навигационная панель — одна из важнейших составных частей любого сайта. Так, в верхней части всех страниц сайта CNN.com есть горизонтальная панель со ссылками на разные новостные рубрики: Технологии, Здоровье, Спорт и т. д. На страницах поисковой выдачи Google имеется аналогичная панель, с помощью которой можно переадресовать поисковый запрос вспомогательным сервисам Google: Картинки, Видео, Карты и пр. На странице, с которой мы экспериментируем, навигационная область в составе верхнего колонтитула содержит ссылки на разные рубрики нашего гипотетического сайта: главную страницу, блог, галерею фотографий и сведения об авторе.

Исходно верстка навигационной панели такова:

```
<div id="nav">
  <ul>
    <li><a href="#">Главная страница</a></li>
    <li><a href="#">Блог</a></li>
    <li><a href="#">Фотоальбом</a></li>
    <li><a href="#">Обо мне</a></li>
  </ul>
</div>
```

Это, как уже нетрудно догадаться, валидный HTML5. Но ничто не говорит клиентской программе, что данный список из четырех элементов — часть навигационного содержимого сайта. Внешне об этом свидетельствуют месторасположение (верхний колонтитул) и текст ссылок, но семантически этот список ссылок ничем не отличается от любого другого.

Для кого же столь важна семантическая верстка сайтовой навигации? Прежде всего для людей с ограниченными возможностями (http://diveintoaccessibility.org). Рассмотрим следующую ситуацию. Пусть, например, вы ограничены в движениях и не можете или с трудом можете пользоваться мышью. Тогда, по-видимому, ради удобства вы установите браузерное расширение, упрощающее переход к важнейшим навигационным ссылкам. Или другой случай: допустим, у вас слабое зрение и вы пользуетесь специальной программой экранного доступа, которая, преобразуя печатный текст в речь, «проговаривает» ключевую информацию веб-страниц. Второй после заголовка (по порядку чтения) особенно информативный элемент страницы — это блок основных навигационных ссылок. Вы, вероятно, настроите программу так, чтобы она после заголовка зачитывала содержимое навигационной панели или (если в этом нет необходимости) сразу переходила к основному содержанию страницы, которое обычно следует за навигационной панелью. Так или иначе возможность программно определять, какая группа ссылок представляет собой навигационный блок, очень важна.

Как видите, применение `<div id="nav">` не считается ошибкой, но вместе с тем оно и не слишком верно в современных условиях. В HTML5 на благо множества людей появился тег верстки навигационных блоков `<nav>`:

```
<nav>
  <ul>
    <li><a href="#">Главная страница</a></li>
    <li><a href="#">Блог</a></li>
```

```
    <li><a href="#">Фотоальбом</a></li>
    <li><a href="#">Обо мне</a></li>
  </ul>
</nav>
```

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Совместимы ли skip-ссылки (skip links) с тегом <nav>? Сохраняется ли потребность в них в HTML5?

Ответ: Skip-ссылки позволяют читателю пролистывать навигационные блоки. Люди с ограниченными возможностями, использующие стороннее ПО для чтения веб-страниц и навигации без мыши, считают skip-ссылки очень полезными. Подробнее о том, зачем и как оформлять ссылки этим способом, читайте на сайте http://www.webaim.org/techniques/skipnav. Как только программы экранного доступа начнут распознавать тег <nav>, потребность в skip-ссылках отпадет: программа-диктор будет сама предлагать пользователю пропустить навигационный блок, обернутый контейнером <nav>. Впрочем, не следует ожидать, что все пользователи с ограниченными возможностями в ближайшем будущем перейдут на программы с поддержкой HTML5-тегов, так что пока для пролистывания <nav>-рубрик следует продолжать указывать skip-ссылки.

Нижние колонтитулы

Наконец мы добрались до конца страницы. Последний в документе контейнер — нижний колонтитул — и есть последнее, о чем мне хочется здесь сказать. Первоначально нижний колонтитул был сверстан так:

```
<div id="footer">
  <p>&#167;</p>
  <p>&#169; 2001&#8211;9 <a href="#">Марк Пилгрим</a></p>
</div>
```

Это валидный HTML5, и можно было бы оставить код таким, если бы в HTML5 не появился специальный тег <footer>:

```
<footer>
  <p>&#167;</p>
  <p>&#169; 2001&#8211;9 <a href="#">Марк Пилгрим</a></p>
</footer>
```

Что следует помещать внутрь контейнера <footer>? Видимо, все то же, что сейчас веб-мастера оформляют с помощью <div id="footer">. Этот уклончивый ответ приводит к порочному кругу. Обратимся к спецификации HTML5, которая гласит: «Нижний колонтитул, как правило, содержит информацию о разделе: указание авторства, ссылки на родственные документы, сведения об авторских правах и др.». В нашем примере все именно так: короткое уведомление о защите прав и ссылка на страницу Об авторе. В том, как велики потенциальные возможности нижних колонтитулов, можно убедиться, глядя на страницы популярных сайтов.

- Нижний колонтитул сайта CNN содержит уведомление о защите прав, ссылки на переводные версии сайта, ссылки на соглашение об использовании, страни-

цу сведений о конфиденциальности, страницы О нас, Связаться и Справка. Весь этот материал вполне отвечает назначению тега <footer>.

- В нижней части главной страницы Google, которая известна своей неоднородностью, есть ссылки на страницы Решения для предприятий, Все о Google, уведомление о защите прав и ссылка на страницу политики конфиденциальности. Все это тоже можно собрать в единый <footer>.
- Нижний колонтитул моего блога (http://diveintomark.org) содержит ссылки на другие мои сайты и уведомление о защите прав — идеальный вариант для <footer>. Стоит отметить, что ссылки не следует помещать внутрь тега <nav>, потому что в данном случае это не часть внутрисайтовой навигации, а всего лишь собрание ссылок на другие мои проекты на сторонних сайтах.

«Тяжелые» нижние колонтитулы (http://ui-patterns.com/pattern/FatFooter) в наши дни приобрели большую популярность. Взгляните, например, на нижний колонтитул сайта W3C (http://www.w3.org). В нем три столбца: Navigation (Навигация), Contact W3C (Связь с W3C) и W3C Updates (Новости W3C). Этот колонтитул сверстан приблизительно так (текст передается в русском переводе):

```
<div id="w3c_footer">
  <div class="w3c_footer-nav">
    <h3>Навигация</h3>
    <ul>
      <li><a href="/">Главная страница</a></li>
      <li><a href="/standards/">Стандарты</a></li>
      <li><a href="/participate/">Участие</a></li>
      <li><a href="/Consortium/membership">Членство</a></li>
      <li><a href="/Consortium/">О Консорциуме</a></li>
    </ul>
  </div>
  <div class="w3c_footer-nav">
    <h3>Связь с W3C</h3>
    <ul>
      <li><a href="/Consortium/contact">Связаться</a></li>
      <li><a href="/Help/">Справка и ЧаВо</a></li>
      <li><a href="/Consortium/sup">Финансовая помощь</a></li>
      <li><a href="/Consortium/siteindex">Карта сайта</a></li>
    </ul>
  </div>
  <div class="w3c_footer-nav">
    <h3>Новости W3C</h3>
    <ul>
      <li><a href="http://twitter.com/W3C">Мы в Twitter</a></li>
      <li><a href="http://identi.ca/w3c">Мы в Identi.ca</a></li>
    </ul>
  </div>
  <p class="copyright">Copyright © 2009 W3C</p>
</div>
```

Для преобразования этого кода в семантический HTML5 надо, на мой взгляд, внести следующую правку.

1. Вместо самого внешнего контейнера <div id="w3c_footer"> использовать тег <footer>.

2. Первые два тега <div class="w3c_footer-nav"> заменить тегом <nav>, а третий — <section>.
3. Вместо заголовков <h3> применить <h1>, потому что каждый из них теперь находится внутри раздела (узел в кратком содержании документа создает как <article>, так и <nav>: см. раздел «Рубрикация» этой главы).

   Итоговый вариант разметки нижнего колонтитула мог бы быть таким:

```
<footer>
  <nav>
    <h1>Навигация</h1>
    <ul>
      <li><a href="/">Главная страница</a></li>
      <li><a href="/standards/">Стандарты</a></li>
      <li><a href="/participate/">Участие</a></li>
      <li><a href="/Consortium/membership">Членство</a></li>
      <li><a href="/Consortium/">О Консорциуме</a></li>
    </ul>
  </nav>
  <nav>
    <h1>Связь с W3C</h1>
    <ul>
      <li><a href="/Consortium/contact">Связаться</a></li>
      <li><a href="/Help/">Справка и ЧаВо</a></li>
      <li><a href="/Consortium/sup">Финансовая помощь</a></li>
      <li><a href="/Consortium/siteindex">Карта сайта</a></li>
    </ul>
  </nav>
  <section>
    <h1>Новости W3C</h1>
    <ul>
      <li><a href="http://twitter.com/W3C">Мы в Twitter</a></li>
      <li><a href="http://identi.ca/w3c">Мы в Identi.ca</a></li>
    </ul>
  </section>
  <p class="copyright">Copyright © 2009 W3C</p>
</footer>
```

Для дальнейшего изучения

Используемые в этой главе страницы-образцы:

- исходная (в разметке HTML 4): http://diveintohtml5.org/examples/blog-original.html;
- измененная (в HTML5-разметке): http://diveintohtml5.org/examples/blog-html5.html.

  Подробнее о кодировках символов:

- «Абсолютный минимум того, что каждый без исключения программист обязан знать о Unicode и кодовых таблицах» (http://www.joelonsoftware.com/articles/Unicode.html) — статья Джоэля Спольского (Joel Spolsky);

- «О ценности Unicode» (http://www.tbray.org/ongoing/When/200x/2003/04/06/Unicode), «О символьных строках» (http://www.tbray.org/ongoing/When/200x/2003/04/13/Strings), «Символы и байты» (http://www.tbray.org/ongoing/When/200x/2003/04/26/UTF) — статьи Тима Брея (Tim Bray).

Подробнее о том, как заставить работать HTML5-теги в Internet Explorer:

- «Применение стилей к элементам, с которыми не знаком IE» (http://xopus.com/devblog/2008/style-unknown-elements.html) — статья Сьерда Вишера (Sjoerd Visscher);
- HTML5 shiv (http://ejohn.org/blog/html5-shiv/) — проект Джона Ресига (John Resig);
- сценарий, включающий поддержку HTML5 (http://remysharp.com/2009/01/07/html5-enabling-script/), — разработка Реми Шарпа (Remy Sharp).

Подробнее о стандартном режиме и ОТД: «ОТД и переключение режимов браузера» (http://hsivonen.iki.fi/doctype/) — статья Генри Сивонена. В данном предмете следует ориентироваться только на сведения этой статьи. Остальные публикации родственной тематики либо неточны, либо неполны, либо устарели.

Валидатор (X)HTML5 вы можете найти по адресу http://html5.validator.nu.

4 С чистого листа (холста)

Приступим

В HTML5 тег <canvas> (http://bit.ly/9JHzOf) определен как «холст для зависимой от разрешения растровой графики, с помощью которого могут на лету прорисовываться диаграммы, графика игр и прочие изображения». На странице холст имеет вид прямоугольника, в границах которого можно рисовать с помощью JavaScript. Во время написания этой книги базовые возможности холста поддерживались в браузерах, перечисленных в табл. 4.1.

Таблица 4.1. Браузеры, в которых поддерживаются возможности холста

IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* В Internet Explorer поддержку обеспечивает сторонняя библиотека ExplorerCanvas.

Как же выглядит холст? А никак, собственно. У тега <canvas> нет своего содержимого и границ. Пример его кода:

```
<canvas width="300" height="225"></canvas>
```

Этот холст показан на рис. 4.1. Граница помечена точками, чтобы наглядно показать его положение.

На одной странице может быть несколько тегов <canvas>. Каждому из них будет соответствовать DOM-объект, и состояние разных холстов независимо. Если приписать каждому из них атрибут id, то для доступа к холстам с помощью JavaScript можно использовать обычный способ.

Включим атрибут id в показанный выше код:

```
<canvas id="a" width="300" height="225"></canvas>
```

Теперь данный тег <canvas> легко обнаруживается в DOM:

```
var a_canvas = document.getElementById("a");
```



Рис. 4.1. Холст с границей

Простые фигуры

В табл. 4.2 перечислены браузеры, в которых поддерживаются простые фигуры.

Таблица 4.2. Браузеры с поддержкой простых фигур

IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* В Internet Explorer поддержку обеспечивает сторонняя библиотека ExplorerCanvas.

По умолчанию холст пуст. Не порисовать ли нам что-нибудь на нем? Для этого с помощью обработчика onclick можно вызвать, к примеру, следующую функцию, которая рисует прямоугольник (интерактивный образец ее работы смотрите на сайте http://diveintohtml5.org/canvas.html):

```
function draw_b() {
  var b_canvas = document.getElementById("b");
  var b_context = b_canvas.getContext("2d");
  b_context.fillRect(50, 25, 150, 100);
}
```

Первая строка в коде этой функции просто находит тег <canvas> в DOM. Интереснее вторая строка. У каждого холста есть «контекст рисования», в котором и происходят все дальнейшие операции. После того как тег <canvas> обнаружен в DOM (методом document.getElementById() или каким-либо другим методом), можно вызвать его метод getContext(), которому следует передать строку "2d":

```
function draw_b() {
  var b_canvas = document.getElementById("b");
  var b_context = b_canvas.getContext("2d");
  b_context.fillRect(50, 25, 150, 100);
}
```

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Существует ли трехмерный холст?

Ответ: Пока еще нет. Отдельные разработчики экспериментировали с собственными API трехмерного рисования, но ни один из этих интерфейсов не стандартизован. Спецификация HTML5 отмечает: «В следующей версии спецификации, вероятно, будет определен контекст 3d».

Итак, даны тег `<canvas>` и его контекст рисования. В этом контексте определены все методы и свойства, отвечающие за рисование. Так, для черчения прямоугольников имеется целая группа свойств и методов.

- Свойство `fillStyle` задает CSS-цвет, узор или градиент (подробнее о градиентах — далее в этой главе). Значение `fillStyle` по умолчанию — непрозрачный черный цвет. Программист может определить какой угодно стиль заливки.
- Метод `fillRect(`*`x, y, width, height`*`)` рисует прямоугольник с текущим стилем заливки.
- Свойство `strokeStyle`, как и `fillStyle`, может задавать CSS-цвет, узор или градиент. Это стиль границ прямоугольника.
- Метод `strokeRect(`*`x, y, width, height`*`)` рисует прямоугольник с текущим стилем границ. При вызове `strokeRect` содержимое прямоугольника не заполняется.
- Метод `clearRect(`*`x, y, width, height`*`)` очищает пикселы внутри заданного прямоугольника.

Каждый контекст рисования «помнит» определенные в нем свойства, пока страница открыта и данные не стираются после какой-либо операции.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Можно ли «перезагрузить» холст?

Ответ: Да. Если переназначить ширину или высоту тега <canvas>, его содержимое будет стерто, а все свойства контекста рисования примут первоначальные значения. Можно даже не менять ширину, а вновь присвоить ей текущее значение, например, так:

```
var b_canvas = document.getElementById("b");
b_canvas.width = b_canvas.width;
```

Вернемся к коду предыдущего образца:

```
var b_canvas = document.getElementById("b");
var b_context = b_canvas.getContext("2d");
b_context.fillRect(50, 25, 150, 100);
```

Вызвав метод `fillRect()`, мы рисуем прямоугольник и заполняем его заливкой текущего стиля. Поскольку значение по умолчанию не менялось, это будет черный цвет. Прямоугольник описывается координатами левого верхнего угла `(50, 25)`, шириной (`150`) и высотой (`100`). Для лучшего усвоения этого механизма рассмотрим координатную систему холста.

Координатная сетка холста

Холст — это двухмерная сетка, на которой координата (0, 0) присвоена левому верхнему углу. Значения абсцисс *(x)* растут слева направо, значения ординат *(y)* — сверху вниз.

Координатная диаграмма на рис. 4.2 изображена с помощью тега <canvas>. В ее состав входят:

- множество полупрозрачных вертикальных линий;
- множество полупрозрачных горизонтальных линий;
- две черные горизонтальные линии;
- две маленькие черные диагональные линии, формирующие стрелку;
- две черные вертикальные линии;
- еще две маленькие черные диагональные линии — тоже стрелка;
- буква x;
- буква y;
- текст (0, 0) возле верхнего левого угла;
- текст (500, 375) возле нижнего правого угла;
- точки в левом верхнем и правом нижнем углах.

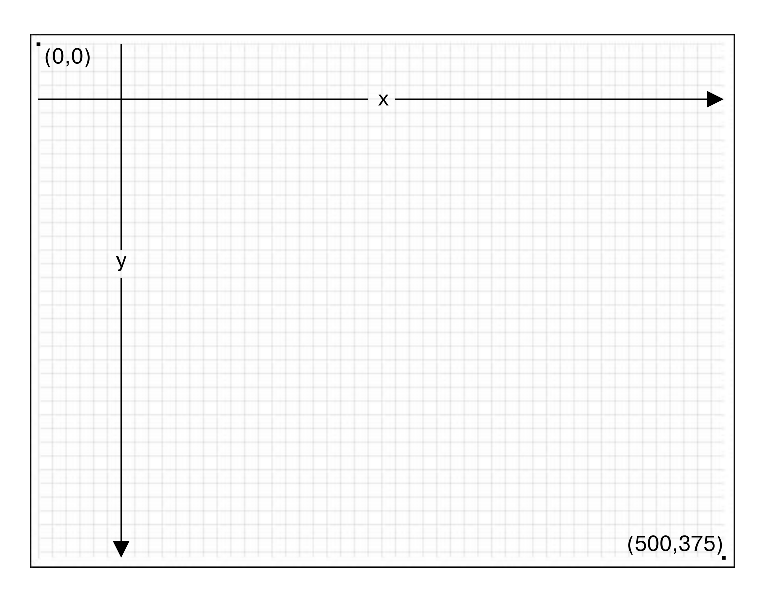


Рис. 4.2. Диаграмма координат холста

Рассмотрим, как изобразить такую составную фигуру.

Сначала нужно задать собственно тег <canvas>. В нем будут определены ширина и высота прямоугольника, а также атрибут id:

```
<canvas id="c" width="500" height="375"></canvas>
```

Затем надо найти в DOM тег <canvas> и получить доступ к его контексту рисования:

```
var c_canvas = document.getElementById("c");
var context = c_canvas.getContext("2d");
```

Только после этого следует приступить к рисованию линий.

Контуры

В табл. 4.3 перечислены браузеры, в которых поддерживаются контуры.

Таблица 4.3. Браузеры с поддержкой контуров

IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* В Internet Explorer поддержку обеспечивает сторонняя библиотека ExplorerCanvas.

Вспомните, как выполняются чертежи тушью. В такой работе спешка вредна: чтобы не сделать ошибку, не чертят сразу набело, а сначала проводят прямые и кривые линии карандашом. Только когда набросок доведен до приемлемого вида, карандашные линии прочерчиваются тушью.

Для каждого холста определен контур. Формирование контура подобно карандашному эскизу: рисовать можно что угодно, но частью конечного продукта наш «эскиз» не станет до тех пор, пока мы не прорисуем его условными «чернилами».

Рисовать прямые линии «карандашом» позволяют следующие два метода:

- moveTo(*x*, *y*) передвигает «карандаш» в заданную начальную точку;
- lineTo(*x*, *y*) проводит линию до заданной конечной точки.

При вызовах moveTo() и lineTo() от раза к разу контур все увеличивается. Эти методы по сути «карандашные», хотя вообще их можно называть как угодно. На холсте ничего не появится, если не вызвать один из «чернильных» методов.

Сформируем сначала координатную сетку:

```
for (var x = 0.5; x < 500; x += 10) {
  context.moveTo(x, 0);
  context.lineTo(x, 375);
}
for (var y = 0.5; y < 375; y += 10) {
  context.moveTo(0, y);
  context.lineTo(500, y);
}
```

До сих пор вызывались только «карандашные» методы и на самом холсте ничего не появилось. Чтобы закрепить эскиз, требуются «чернила»:

```
context.strokeStyle = "#eee";
context.stroke();
```

Метод stroke() — один из «чернильных» методов. Он принимает тот сложный контур, который программист определил с помощью moveTo() и lineTo(), и прори-

совывает этот контур на холсте. Цвет линий определяет свойство `strokeStyle`. Результат показан на рис. 4.3.

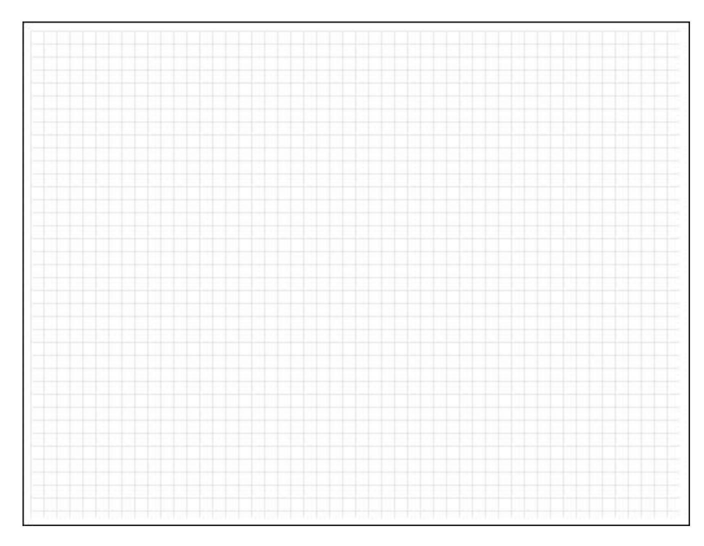


Рис. 4.3. Прямоугольная сетка на холсте

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Почему начальным значением для x и y выбрано 0,5, а не 0?

Ответ: Представьте себе каждый отдельный пиксел как большой квадрат. Целочисленная координатная сетка (0, 1, 2...) проходит по стыкам квадратов. Если провести линию единичной толщины по такому стыку, то она захватит края прилегающих квадратов и окажется толщиной два пиксела. Чтобы добиться толщины ровно 1 пиксел, надо сдвинуть координату на 0,5 в направлении, перпендикулярном направлению линии.

Теперь изобразим горизонтальную стрелку. Все прямые и кривые участки одного контура изображаются одним и тем же цветом (или градиентом, о чем уже совсем скоро пойдет речь). Но для стрелки нужен другой цвет: черный, а не светло-серый. Поэтому начнем новый контур:

```
context.beginPath();
context.moveTo(0, 40);
context.lineTo(240, 40);
context.moveTo(260, 40);
context.lineTo(500, 40);
context.moveTo(495, 35);
context.lineTo(500, 40);
context.lineTo(495, 45);
```

Аналогично создается вертикальная стрелка. Поскольку она того же цвета, что и горизонтальная, начинать новый контур *нет* необходимости. Две стрелки будут частями единого контура:

```
context.moveTo(60, 0);
context.lineTo(60, 153);
context.moveTo(60, 173);
context.lineTo(60, 375);
context.moveTo(65, 370);
context.lineTo(60, 375);
context.lineTo(55, 370);
```

Как я сказал, стрелки будут черного цвета, но свойство strokeStyle пока еще определяет светло-серый цвет (при создании нового контура значения fillStyle и strokeStyle не возвращаются к начальным). Это не составляет проблемы, потому что до сих пор вызывались только «карандашные» методы. Прежде чем прорисовывать контур по-настоящему, «чернилами», зададим в свойстве strokeStyle значение черного цвета, иначе стрелки окажутся почти невидимыми:

```
context.strokeStyle = "#000";
context.stroke();
```

Результат можно видеть на рис. 4.4.

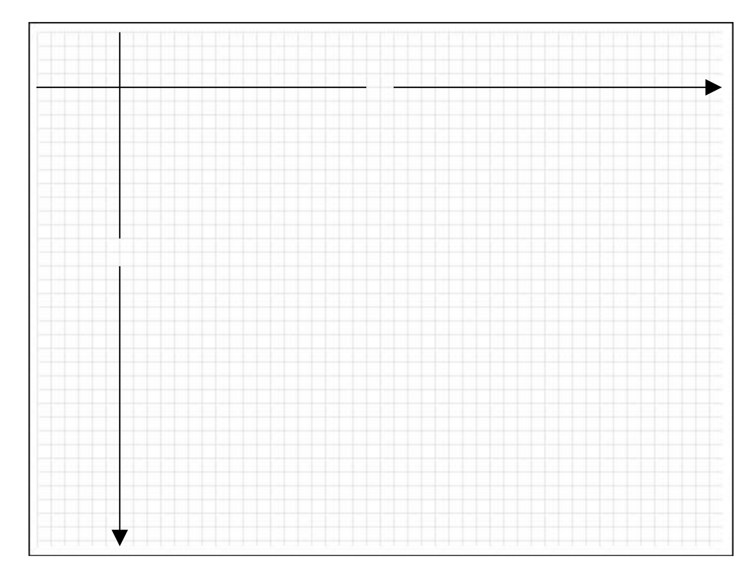


Рис. 4.4. Координатные оси (без ярлыков) на холсте

Текст

На холсте можно рисовать не только линии, но и текст (табл. 4.4). В отличие от окружающего текста веб-страницы, текст на холсте не подчиняется «блочной» модели, то есть к нему неприменимы известные технологии CSS-форматирования: плавающая разметка, поля, отступы и перенос по словам (это и к лучшему, как вы могли бы подумать). Можно установить некоторые атрибуты шрифта, после чего выбирается точка на холсте и прорисовывается текст.

Таблица 4.4. Браузеры, в которых поддерживается текст на холсте

IE*	Firefox**	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* В Internet Explorer поддержку обеспечивает сторонняя библиотека ExplorerCanvas.

** В Mozilla Firefox 3.0 для обеспечения поддержки нужен пакет совместимости.

В контексте рисования (см. раздел «Простые фигуры» этой главы) доступны следующие атрибуты шрифта:

- `font` — все, что позволяет определить одноименное CSS-правило: стиль, вариант, начертание, размер, интерлиньяж и семейство шрифтов;
- `textAlign` — задает выравнивание текста, подобно CSS-правилу `text-align`, но не точно так же; возможные значения этого свойства: `start`, `end`, `left`, `right`, `center`;
- `textBaseline` — определяет положение текста относительно начальной точки; возможные значения этого свойства: `top`, `hanging`, `middle`, `alphabetic`, `ideographic`, `bottom`.

Свойство `textBaseline` непростое, потому что непрост и сам текст (английский текст, как и русский, достаточно прост, но на холст можно вывести любой символ Unicode, а там простота заканчивается). Спецификация HTML5 так описывает различные базовые линии текста[1]:

> Верх и низ em-квадрата — это приблизительные верхняя и нижняя границы символов шрифта. Строго посередине между ними проводится средняя линия. К «висячей» базовой линии прикрепляются такие символы, как आ; к «алфавитной» базовой линии — такие символы, как Á, ÿ, f и Ω; к «идеографической» базовой линии — такие символы, как 私 и 達. По вине символов, далеко выходящих за границы em-квадратов (рис. 4.5), верхний и нижний края ограничивающего блока могут находиться далеко от базовых линий.



Рис. 4.5. Базовые линии текста

Для таких простых алфавитов, как английский и русский, можно ограничиться значениями `top`, `middle` и `bottom` свойства `textBaseline`.

[1] http://bit.ly/aHCdDO. — *Примеч. авт.*

Итак, нарисуем какой-нибудь текст. Текст на холсте наследует размер шрифта и стиль, применяемые к самому тегу <canvas>, но эти значения можно переопределить в контексте рисования с помощью свойства font:

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

Прорисовка текста выполняется методом fillText():

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Можно ли при рисовании текста использовать относительный размер шрифта?

Ответ: Да. Для тега <canvas>, как и для любого другого HTML-тега на странице, размер шрифта вычисляется исходя из таблицы стилей. Если присвоить context.font относительное значение, например 1.5em или 150%, то браузер умножит эту величину на размер шрифта, вычисленный им для тега <canvas>.

Пусть, например, мы хотим, чтобы текст в левом верхнем углу располагался под линией y = 5. Из обычной программистской лености не станем мерить высоту текста, чтобы рассчитать ординату базовой линии, а вместо этого присвоим textBaseline значение top и передадим методу-обработчику координаты левого верхнего угла рамки, ограничивающей текст:

```
context.textBaseline = "top";
context.fillText("( 0 , 0 )", 8, 5);
```

Теперь рассмотрим текст в правом нижнем углу. Пусть, например, нам угодно, чтобы правый нижний угол области, занятой этим текстом, имел координаты (492, 370), то есть располагался всего в нескольких пикселах от правого нижнего угла холста. Опять же мы не будем измерять ширину и высоту текста. Достаточно установить textAlign равным right и textBaseline равным bottom, а затем вызвать fillText(), передав ему координаты правого нижнего угла рамки, ограничивающей текст:

```
context.textAlign = "right";
context.textBaseline = "bottom";
context.fillText("( 500 , 375 )", 492, 370);
```

Результат этих манипуляций показан на рис. 4.6.

Как же мы забыли о точках по углам? Рисование кругов будет рассмотрено чуть позже, а пока что мы схитрим и изобразим точки в виде прямоугольников (см. раздел «Простые фигуры» этой главы):

```
context.fillRect(0, 0, 3, 3);
context.fillRect(497, 372, 3, 3);
```

Вот и все! Итог наших операций показан на рис. 4.7.

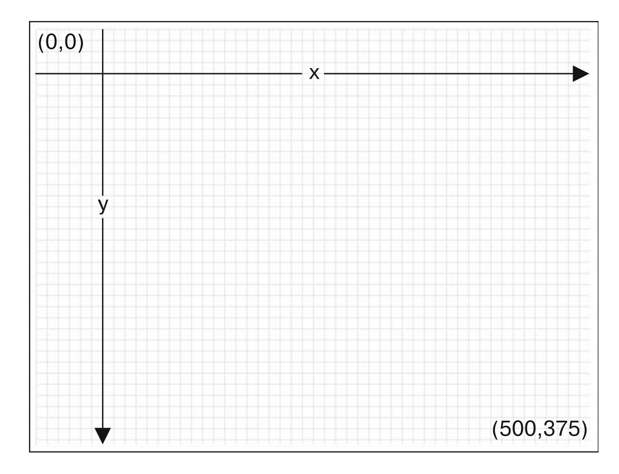


Рис. 4.6. Координатные оси с ярлыками на холсте

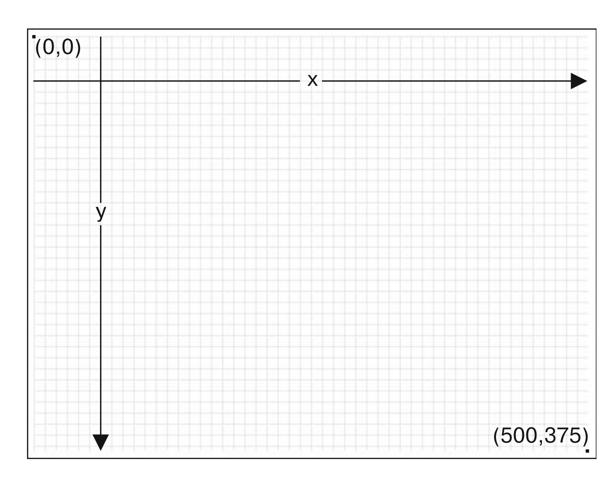


Рис. 4.7. Диаграмма координат холста, созданная с помощью холста

Градиенты

В табл. 4.5 перечислены браузеры, в которых поддерживаются градиенты.

Таблица 4.5. Браузеры с поддержкой градиентов

	IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
Линейные градиенты	7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
Радиальные градиенты	–	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* В Internet Explorer поддержку обеспечивает сторонняя библиотека ExplorerCanvas.

В этой главе вы сначала научились рисовать прямоугольники с непрозрачной цветовой заливкой (см. раздел «Простые фигуры» этой главы), а потом линии — тоже непрозрачным цветом (см. раздел «Контуры» этой главы). Но для фигур и линий пригодны не только сплошные цвета. Существует градиентная заливка, позволяющая добиваться замечательного внешнего эффекта. Примером послужит рис. 4.8.

Этот холст верстается, как и всякий другой:

```
<canvas id="d" width="300" height="225"></canvas>
```

Сценарий должен сначала обнаружить тег `<canvas>` и получить доступ к его контексту рисования:

```
var d_canvas = document.getElementById("d");
var context = d_canvas.getContext("2d");
```



Рис. 4.8. Линейный градиент в направлении слева направо

Войдя в контекст рисования, мы можем начать задавать градиент. Градиентом называется плавный переход между двумя или несколькими цветами. Контекст рисования HTML5-холста поддерживает два типа градиентов.

- `createLinearGradient(`*`x0, y0, x1, y1`*`)` — направляет заливку вдоль отрезка из точки (*x0, y0*) в точку (*x1, y1*).
- `createRadialGradient(`*`x0, y0, r0, x1, y1, r1`*`)` — направляет заливку вдоль конуса, образуемого двумя окружностями. Начальные три параметра описывают первую окружность с центром (*x0, y0*) и радиусом *r0*. Конечные три параметра описывают вторую окружность: центр (*x1, y1*) и радиус *r1*.

Создадим линейный градиент. Градиенты бывают любых размеров; наш будет шириной 300 пикселов, как и весь холст:

```
var my_gradient = context.createLinearGradient(0, 0, 300, 0);
```

Поскольку значения ординат (второй и четвертый параметры) равны `0`, эта градиентная заливка будет равномерным переходом от одного цвета к другому в направлении слева направо.

Имея объект-градиент, можно задать его цвета. Линейный градиент должен иметь две или несколько цветовых точек, расположенных в любых местах на его

протяжении. Чтобы добавить цветовую точку, укажите ее позицию на градиенте, пользуясь числами из интервала от 0 до 1.

Зададим плавное «выцветание» из черного цвета в белый:

```
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
```

На холсте ничего не изображено, хотя градиентная заливка уже определена: это пока лишь объект в памяти. Чтобы нарисовать градиент, присвойте `fillStyle` соответствующее значение и нарисуйте фигуру, например прямоугольник:

```
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Результат будет таким же, как на рис. 4.8.

А если бы вы захотели создать градиентную заливку с переходом от одного цвета к другому в направлении сверху вниз? Тогда постоянными должны быть значения абсцисс (первый и третий параметры), а значения ординат должны ограничивать диапазон от 0 до высоты холста:

```
var my_gradient = context.createLinearGradient(0, 0, 0, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Результат показан на рис. 4.9.

Можно также прорисовать градиент вдоль диагонали, например:

```
var my_gradient = context.createLinearGradient(0, 0, 300, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Результат показан на рис. 4.10.



Рис. 4.9. Линейный градиент в направлении сверху вниз



Рис. 4.10. Диагональный линейный градиент

Изображения

В табл. 4.6 перечислены браузеры с поддержкой изображений на холсте.

Таблица 4.6. Браузеры, в которых поддерживаются картинки на холсте

IE*	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

* В Internet Explorer поддержку обеспечивает сторонняя библиотека ExplorerCanvas.

На рис. 4.11 показан кот, отображаемый с помощью тега `<img>`.

На рис. 4.12 показан тот же самый кот, нарисованный на HTML5-холсте.



Рис. 4.11. Кот в теге <img>



Рис. 4.12. Кот в теге <canvas>

В контексте рисования определено несколько методов прорисовки графических файлов на холсте:

- `drawImage(image, dx, dy)` — просто берет картинку и рисует ее; координаты `(dx, dy)` указывают на верхний левый угол изображения, то есть если присвоить им значение `(0, 0)`, то картинка будет помещена в верхний левый угол холста;
- `drawImage(image, dx, dy, dw, dh)` — берет картинку, масштабирует ее по горизонтали и вертикали, чтобы она соответствовала заданным ширине `dw` и высоте `dh`, после чего рисует ее от точки `(dx, dy)`;
- `drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)` — берет картинку, выделяет на ней прямоугольник `(sx, sy, sw, sh)`, масштабирует его до размеров `(dw, dh)` и рисует от точки `(dx, dy)`.

Параметры метода `drawImage()` так объяснены в спецификации HTML5 (http://bit.ly/9WTZAp):

> Начальный (source) прямоугольник — это прямоугольная область [на исходной картинке], углы которой имеют координаты (sx, sy), (sx+sw, sy), (sx+sw, sy+sh), (sx, sy+sh).
> Конечный (destination) прямоугольник — это прямоугольная область [на холсте], углы которой имеют координаты (dx, dy), (dx+dw, dy), (dx+dw, dy+dh), (dx, dy+dh).

Наглядно эти параметры показаны на рис. 4.13.

Чтобы нарисовать картинку на холсте, нужна собственно картинка, которая была бы размещена в существующем теге `<img>` или представлена JavaScript-объектом `Image`. Во всяком случае, прежде чем переходить к рисованию, убедитесь, что картинка полностью загрузилась.

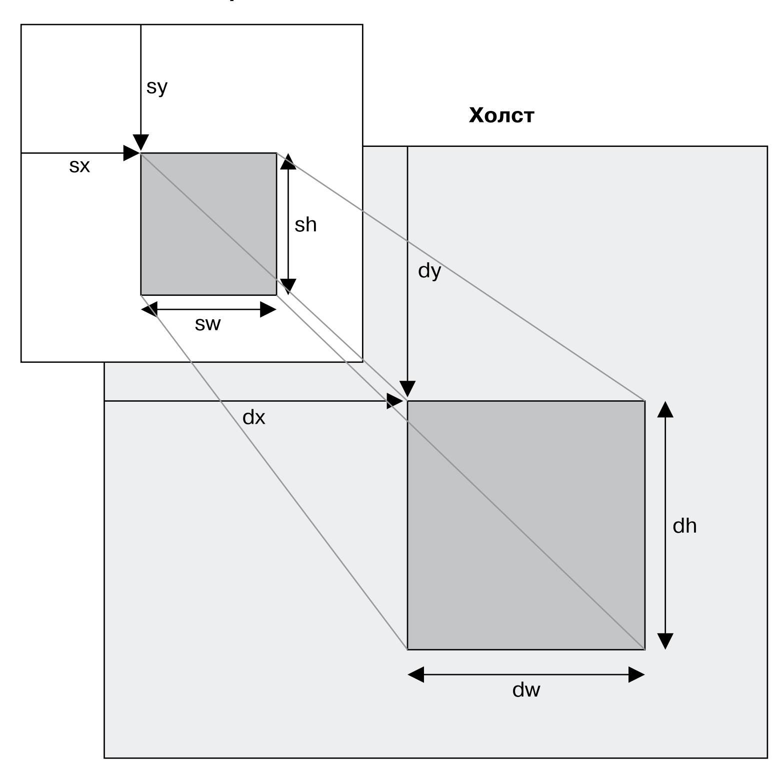


Рис. 4.13. Метод drawImage() переносит изображение на холст

При использовании тега <img> прорисовку на холсте можно вызывать событием window.onload:

```
<img id="cat" src="images/cat.png" alt="спящий котик" width="177" height="113">
<canvas id="e" width="177" height="113"></canvas>
<script>
  window.onload = function() {
    var canvas = document.getElementById("e");
    var context = canvas.getContext("2d");
    var cat = document.getElementById("cat");
    context.drawImage(cat, 0, 0);
  };
</script>
```

Если же вы работаете с картинкой — объектом JavaScript, то уместно будет вызывать прорисовку по наступлении события Image.onload:

```
<canvas id="e" width="177" height="113"></canvas>
<script>
  var canvas = document.getElementById("e");
  var context = canvas.getContext("2d");
  var cat = new Image();
```

```
  cat.src = "images/cat.png";
  cat.onload = function() {
    context.drawImage(cat, 0, 0);
  };
</script>
```

Необязательные третий и четвертый параметры метода drawImage() отвечают за масштабирование картинки. На рис. 4.14 показано то же самое изображение кота, уменьшенное вдвое по ширине и высоте и отрисованное от разных точек единого холста.

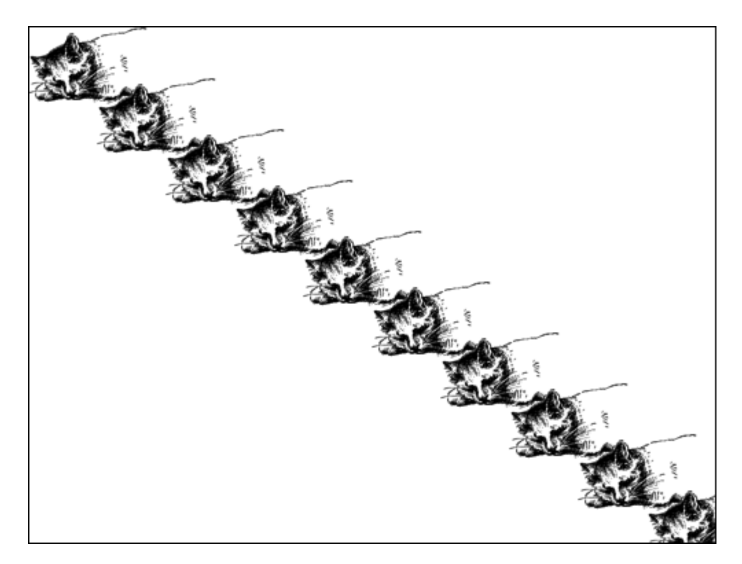


Рис. 4.14. Много котиков!

Вот сценарий, с помощью которого создан эффект «многокотья»:

```
cat.onload = function() {
  for (var x = 0, y = 0;
    x < 500 && y < 375;
    x += 50, y += 37) {
    context.drawImage(cat, x, y, 88, 56);
  }
};
```

После всех этих операций встает закономерный вопрос: зачем вообще рисовать картинки на холсте? Достигаем ли мы (ценой дополнительной сложности) хоть каких-нибудь преимуществ по сравнению с <img> и некоторыми CSS-правилами? Ведь даже замечательное «многокотье» можно создать 10 перекрывающимися тегами <img>.

Зачем, говорите вы? Затем же, зачем нужна возможность рисования текста (см. раздел «Текст» этой главы). В состав диаграммы координат холста (см. раздел «Координатная сетка холста» этой главы) входили текст, линии и фигуры: рисованный текст выступил здесь как часть составной графической работы. Легко

представить себе на более сложных диаграммах значки, спрайты и другие графические детали, отрисованные с помощью drawImage().

А что в IE?

Internet Explorer до версии 8 включительно (она была последней стабильной на момент написания этой книги) не поддерживает API рисования. IE, однако, поддерживает проприетарную технологию Microsoft под названием VML, которая обладает значительной долей функциональности тега <canvas>. Так появился сценарий excanvas.js.

ExplorerCanvas (http://code.google.com/p/explorercanvas/), или excanvas.js, — это JavaScript-библиотека с открытым исходным кодом, распространяемая под лицензией Apache. В ней реализован API рисования для Internet Explorer, воспользоваться которым можно, следующим образом добавив в верхнюю часть страницы тег <script>:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Название моей страницы</title>
    <!--[if IE]>
      <script src="excanvas.js"></script>
    <![endif]-->
  </head>
  <body>
    …
  </body>
</html>
```

Строки <!--[if IE]> и <![endif]--> обрамляют условный комментарий, в котором Internet Explorer видит утверждение: «Если текущий браузер — Internet Explorer какой угодно версии, то исполнить следующий код». Все остальные браузеры обращаются с условным комментарием, как и с любым другим HTML-комментарием. В результате Internet Explorer загрузит и исполнит сценарий excanvas.js, тогда как другой браузер совершенно проигнорирует его (не загрузит и не исполнит). Это ускоряет загрузку страницы в браузерах со встроенной поддержкой API рисования.

После того как вы поместите сценарий excanvas.js в контейнер <head>, больше никаких мер по обеспечению совместимости с Internet Explorer не требуется. Теги <canvas> теперь можно добавлять в код страниц или создавать динамически, с помощью JavaScript. Следуя рекомендациям этой главы, вы сможете входить в контекст рисования и рисовать фигуры, линии, градиенты, текст.

Однако не все так просто. Известно несколько ограничений excanvas.js.

- Градиенты (см. раздел «Градиенты» этой главы) могут быть только линейными. Радиальные градиенты не поддерживаются.
- Узор должен повторяться в обоих направлениях.

- Не поддерживается отрисовка фрагмента картинки.
- Масштабирование не является единообразным, из-за чего линии масштабируются неправильно.
- Замедляется работа. Это, конечно, не должно удивлять читателя хотя бы потому, что JavaScript-парсер у Internet Explorer медленнее, чем у конкурентов. IE может совершенно увязнуть в обработке такой задачи, как рисование сложных форм с помощью JavaScript-библиотеки, которая пересылает команды постороннему механизму. В простых случаях с несколькими линиями и картинками производительность системы почти не упадет, но если вы будете анимировать картинки на холсте и т. п., то сразу почувствуете разницу.

При использовании `excanvas.js` нужно помнить еще об одной проблеме, с которой я столкнулся при разработке иллюстративного материала этой главы. Если веб-мастер поместит сценарий `excanvas.js` на HTML-страницу, то ExplorerCanvas будет инициализировать свой интерфейс, но это, однако, не значит, что Internet Explorer окажется готов к его немедленному использованию. Иногда бывает так, что «лжехолст» (faux-canvas) *почти* подготовлен к работе, но не до конца. Верный симптом этого состояния — когда при любой операции с тегом `<canvas>`, например при попытке войти в его контекст рисования, Internet Explorer жалуется: «Объект не поддерживает это свойство (или этот метод)».

Проще всего во избежание неполадок отложить манипуляции с холстом на время после события `onload`. Если на странице много картинок или видеоклипов, придется подождать: ведь событие `onload` отмечает момент полной загрузки содержимого, зато ExplorerCanvas успеет ввести в бой все свои резервы.

Живой пример

«Уголки» (Halma) — это старинная настольная игра, существующая во множестве вариантов. Для этого раздела я написал пасьянсный (для одного игрока) вариант уголков с девятью фишками на доске 9 × 9. В начале игры фишки образуют квадрат 3 × 3 в левом нижнем углу доски. Цель игры — в наименьшее возможное количество ходов передвинуть фишки так, чтобы они образовали квадрат 3 × 3 в правом верхнем углу доски.

При игре в «Уголки» разрешены два типа ходов.

- Взяв фишку, передвинуть ее на любое прилегающее пустое поле. Пустым называется поле доски, на котором в данный момент нет фишки. Прилегающим считается каждое из тех восьми, которыми непосредственно окружено место нынешнего расположения передвигаемой фишки (противоположные края доски не сомкнуты, поэтому фишка на первой вертикали не может двигаться налево: на «запад», «северо-запад» и «юго-запад»; точно так же фишка на первой горизонтали не может двигаться вниз: на «юг», «юго-запад» и «юго-восток»).
- Взяв фишку, перенести ее через одну или несколько прилегающих фишек. При переносе поверх непустого прилегающего поля перемещение фишки далее, поверх непустых полей, соседствующих с данным, или через них, не считается за отдельный ход, то есть фактически любая непрерывная последовательность

переносов — это один ход. Поскольку цель игры в том, чтобы свести количество ходов к минимуму, хорошие игроки строят длинные цепи фишек, поверх которых затем можно систематически переносить другие фишки.

Скриншот игры показан на рис. 4.15. Можете поиграть в описываемый здесь вариант уголков (http://diveintohtml5.org/examples/canvas-halma.html) и заглянуть в код страницы.

«Как же все это работает?» — спросите вы, и очень кстати. Я не буду полностью приводить здесь код основного сценария игры (см. http://diveintohtml5.org/examples/halma.js). Пропуская большинство функций, отвечающих собственно за игровую часть, я обращу внимание лишь на некоторые фрагменты кода, которые касаются прорисовки на холсте и обработки щелчков кнопкой мыши на теге `<canvas>`.

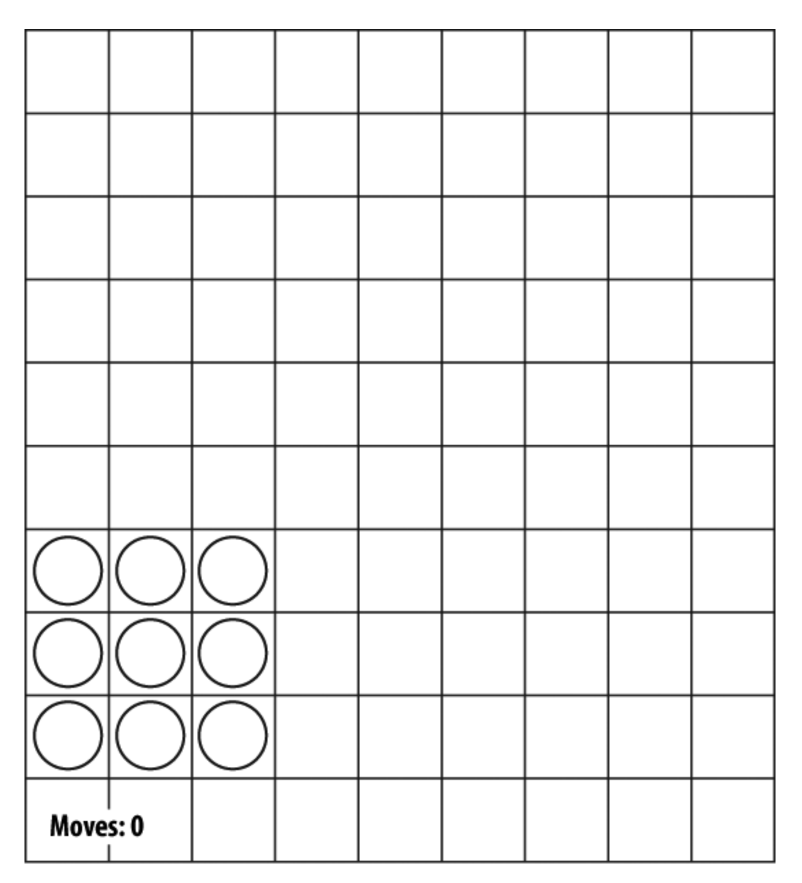


Рис. 4.15. Начальная позиция в игре «Уголки»

Во время загрузки страницы мы инициализируем игру: устанавливаем размеры холста и сохраняем в переменной ссылку на его контекст рисования:

```
gCanvasElement.width = kPixelWidth;
gCanvasElement.height = kPixelHeight;
gDrawingContext = gCanvasElement.getContext("2d");
```

Затем мы делаем нечто совершенно новое, не показанное ранее: добавляем к тегу <canvas> «слушатель событий», который будет реагировать на щелчки кнопкой мыши:

```
gCanvasElement.addEventListener("click", halmaOnClick, false);
```

Таким образом, функция halmaOnClick() вызывается каждый раз, когда указатель мыши находится где-либо на холсте, а пользователь щелкает кнопкой мыши. Ей передается аргумент — объект MouseEvent с данными о положении указателя в момент щелчка:

```
function halmaOnClick(e) {
  var cell = getCursorPosition(e);
  // the rest of this is just gameplay logic
  for (var i = 0; i < gNumPieces; i++) {
    if ((gPieces[i].row == cell.row) &&
        (gPieces[i].column == cell.column)) {
        clickOnPiece(i);
        return;
    }
  }
  clickOnEmptyCell(cell);
}
```

На следующем шаге нужно взять объект MouseEvent и по его данным рассчитать, на какое из полей игровой доски пришелся щелчок. Доска занимает весь холст, поэтому любой щелчок падает на одно из ее полей. Остается только узнать, какое именно это поле. Задача не из легких, ведь в различных браузерах «события мыши» реализованы по-разному:

```
function getCursorPosition(e) {
  var x;
  var y;
  if (e.pageX || e.pageY) {
    x = e.pageX;
    y = e.pageY;
  }
  else {
    x = e.clientX + document.body.scrollLeft +
    document.documentElement.scrollLeft;
    y = e.clientY + document.body.scrollTop +
    document.documentElement.scrollTop;
  }
```

На данном этапе мы имеем координаты x и y, описывающие относительное положение указателя в документе, то есть на всей HTML-странице. Чтобы превратить их в координаты на холсте, сделаем следующее:

```
x -= gCanvasElement.offsetLeft;
y -= gCanvasElement.offsetTop;
```

Теперь у нас есть координаты x и y, отсчитываемые в пикселах от угла холста (см. раздел «Координатная сетка холста» этой главы). Например, если x и y сейчас равны 0, то мы понимаем, что пользователь щелкнул на крайнем левом верхнем пикселе холста.

Рассчитаем наконец, на какое из полей игровой доски пришелся щелчок, чтобы далее система действовала по обстоятельствам:

```
  var cell = new Cell(Math.floor(y/kPieceWidth),
                      Math.floor(x/kPieceHeight));
  return cell;
}
```

Да, в «событиях мыши» разобраться непросто. Но ту же самую логику и, в сущности, тот же самый код вы можете применять во всех своих веб-приложениях на основе холста. Запомните: щелчок кнопкой мыши → координаты относительно документа → координаты относительно холста → внутренняя логика программы.

Сейчас посмотрим на основную графическую процедуру. Поскольку графика игры очень проста, я счел возможным при любом изменении на доске очищать холст и перерисовывать его содержимое. Вообще говоря, необходимости в этом нет: контекст рисования хранит текущие данные даже в том случае, если пользователь прокручивает страницу до полного исчезновения холста из виду или переходит к другим вкладкам (и затем возвращается). В веб-приложениях на основе холста, имеющих более сложную графику (таких как игры-аркады), приходится оптимизировать прорисовку: следить за тем, какие области холста «загрязнились», и перерисовывать только их. Решение подобных задач выходит за пределы данной книги.

Вот код, очищающий доску:

```
gDrawingContext.clearRect(0, 0, kPixelWidth, kPixelHeight);
```

Последовательность команд, отвечающих за прорисовку доски, должна показаться вам знакомой: схожим образом мы рисовали координатную диаграмму холста (см. раздел «Координатная сетка холста» этой главы):

```
gDrawingContext.beginPath();
/* намечаем вертикальные линии */
for (var x = 0; x <= kPixelWidth; x += kPieceWidth) {
  gDrawingContext.moveTo(0.5 + x, 0);
  gDrawingContext.lineTo(0.5 + x, kPixelHeight);
}
/* намечаем горизонтальные линии */
for (var y = 0; y <= kPixelHeight; y += kPieceHeight) {
  gDrawingContext.moveTo(0, 0.5 + y);
  gDrawingContext.lineTo(kPixelWidth, 0.5 + y);
}
/* рисуем! */
gDrawingContext.strokeStyle = "#ccc";
gDrawingContext.stroke();
```

По-настоящему интересно станет, когда мы начнем прорисовывать каждую отдельную фишку. Фишки круглые, а мы до сих пор не рисовали окружностей. Кроме того, если пользователь выбирает фишку перед тем, как совершить ею ход, надо отображать такую фишку как круг с заливкой. Далее аргумент p обозначает фишку (piece) со свойствами row и column, описывающими текущее положение фишки на доске. Внутренние константы игры, то есть номера горизонталей и вертикалей, мы будем преобразовывать в относительные координаты на холсте (x, y), затем будем рисовать окружность и, наконец, если фишка выбрана пользователем, будем заливать круг непрозрачным цветом:

```
function drawPiece(p, selected) {
  var column = p.column;
  var row = p.row;
  var x = (column * kPieceWidth) + (kPieceWidth/2);
  var y = (row * kPieceHeight) + (kPieceHeight/2);
  var radius = (kPieceWidth/2) - (kPieceWidth/10);
```

Специфическая для игры логика на этом заканчивается. Теперь в нашем распоряжении пара координат (x, y) на холсте, обозначающая центр окружности, которую мы хотим провести. В API рисования нет метода circle(), но есть метод arc(). Вы еще не забыли школьную геометрию? Окружность — не что иное, как замкнутая на себе дуга постоянной кривизны. Метод arc() принимает в качестве аргументов координаты (x, y) центра кривизны, величину радиуса, начальный и конечный углы (в радианах), а также указатель направления (false — по часовой стрелке, true — против часовой стрелки). Для расчета значения в радианах воспользуемся встроенным JavaScript-модулем Math:

```
gDrawingContext.beginPath();
gDrawingContext.arc(x, y, radius, 0, Math.PI * 2, false);
gDrawingContext.closePath();
```

Но погодите: на холсте еще ничего не появилось. Метод arc(), подобно moveTo() и lineTo(), «карандашный» (см. раздел «Контуры» этой главы). Для настоящей прорисовки окружности надо выбрать желаемый strokeStyle и вызвать stroke():

```
gDrawingContext.strokeStyle = "#000";
gDrawingContext.stroke();
```

А если фишка выбрана? Используем тот же контур окружности, но на этот раз зальем круг непрозрачным цветом:

```
if (selected) {
  gDrawingContext.fillStyle = "#000";
  gDrawingContext.fill();
}
```

Вот и... нет, не все, но почти все. Остальная часть сценария — функции, специфические для игры в «Уголки». Они определяют, допустим ли тот или иной ход, следят за количеством совершенных ходов и отмечают момент завершения игры. Вот так из девяти кругов, нескольких отрезков, обработчика onclick и HTML5-холста у нас получилась вполне занимательная игра.

Для дальнейшего изучения

- Пособие по использованию холста (https://developer.mozilla.org/en/Canvas_tutorial) на сайте Центра Mozilla для веб-разработчиков.
- «Холст в HTML5: основы» (http://dev.opera.com/articles/view/html-5-canvas-the-basics/) — статья Михая Сукана (Mihai Sucan).
- Образцы применения HTML-тега `<canvas>`, инструменты и руководства (http://www.canvasdemos.com).
- Тег `<canvas>` (http://bit.ly/9JHzOf) в черновике стандарта HTML5.

5 Видео в Сети[1]

Приступим

Любой человек, которому за последние четыре года хотя бы раз довелось посетить YouTube.com, знает, что на веб-страницах можно размещать видеоклипы. Но до появления HTML5 стандартизованного способа делать это не существовало. Едва ли не все видео, которое вам когда-либо приходилось видеть в Сети, достигало ваших глаз через сторонние приложения: QuickTime, RealPlayer или Flash (на YouTube, в частности, применяется Flash). Такие приложения достаточно тесно интегрированы с браузером, так что пользователь может даже не сознавать, что пользуется ими. Попробуйте открыть веб-видео на платформе, не поддерживающей нужное приложение, чтобы понять реальное положение вещей.

HTML5 определяет стандартный способ встраивания видео на веб-страницы — тег `<video>`. Поддержку тега `<video>` в браузерах до сих пор улучшают, то есть, если говорить более прямо, он пока не работает или, во всяком случае, работает не везде. Однако не отчаивайтесь: есть множество альтернатив и компромиссных вариантов.

В табл. 5.1 показано, в каких браузерах на момент написания книги имелась поддержка тега `<video>`.

Таблица 5.1. Поддержка тега <video>

IE9	IE8	IE7	Firefox 3.5	Firefox 3.0	Safari 4	Safari 3	Chrome	Opera
√	–	–	√	–	√	√	√	√

Поддержка тега `<video>` — только часть вопроса. Прежде чем говорить об HTML5-видео, надо усвоить некоторые общие сведения о видеофайлах (если в этой области вы достаточно подготовлены, переходите сразу к разделу «Что работает в Интернете?» этой главы).

Видеоконтейнеры

Говоря о видеофайлах, часто упоминают AVI-файлы, MP4-файлы и т. п. На самом же деле AVI и MP4 — это лишь форматы контейнеров. Как в ZIP-файле может

[1] В этой главе мы используем слово «контейнер» только в значении «видеоконтейнер». В остальных главах оно может употребляться как равнозначное с терминами «элемент» и «тег». — *Примеч. перев.*

помещаться информация любого рода, так и в мультимедийном контейнере определен только *способ* хранения содержимого, а не *тип* этого содержимого (вообще-то ситуация чуть сложнее: совместимы между собой не все видеопотоки и форматы контейнеров, но пока это неважно).

Видеофайл обычно содержит не менее двух дорожек: видеодорожку (без звука) и одну или несколько аудиодорожек (без картинки). Дорожки, как правило, взаимосвязаны посредством маркеров в аудиоданных; с их помощью достигается согласованность звукового ряда со зрительным. Отдельные дорожки могут быть оснащены метаинформацией, такой как соотношение сторон кадра (в видео) или язык (в аудио). Метаданные целого видеофайла: название, графическая заставка, номер эпизода телесериала и т. д. — тоже могут присутствовать в контейнере.

Есть немало форматов видеоконтейнеров. Рассмотрим некоторые из самых популярных.

MPEG-4 — файлы с расширением MP4 и M4V. Контейнер MPEG-4 основан на ранее разработанной Apple технологии контейнеров QuickTime (расширение MOV). Трейлеры фильмов на сайте Apple до сих пор публикуются в этих более старых контейнерах, а фильмы на iTunes распространяются в MPEG-4.

Flash Video — файлы с расширением FLV. Воспроизводятся, естественно, проигрывателем Adobe Flash. В версиях Flash, более ранних, чем 9.0.60.184 (Flash Player 9 с пакетом обновлений 3), поддерживался только этот формат контейнеров. В новых версиях добавилась поддержка MPEG-4.

Ogg — файлы с расширением OGV. Это открытый стандарт, не ограниченный какими-либо патентами и дружественный к open-source-сообществу. Встроенная поддержка формата контейнеров Ogg, Ogg-видеокодека Theora и аудиокодека Vorbis имеется в Firefox 3.5, Chrome 4 и Opera 10.5. Все основные дистрибутивы Linux умеют работать с Ogg сразу после установки, а на десктопных платформах Mac и Windows должны быть дополнительно установлены компоненты QuickTime и фильтры DirectShow соответственно. Воспроизводить Ogg на всех платформах умеет превосходный проигрыватель VLC (http://www.videolan.org/vlc/).

WebM — файлы с расширением WEBM. Это новый формат видеоконтейнеров, технически очень похожий на другой, более ранний формат под названием Matroska («Матрешка»). О создании WebM было объявлено на Конференции разработчиков Google I/O 2010. Его планируется использовать исключительно в связке с видеокодеком VP8 и аудиокодеком Vorbis (о которых подробнее читайте далее). Встроенная поддержка WebM-видео обещает быть в очередных версиях Chromium, Google Chrome, Mozilla Firefox и Opera. Компания Adobe также объявила, что следующая версия Flash будет поддерживать формат WebM.

Audio Video Interleave («чередование аудио и видео») — файлы с расширением AVI. Формат видеоконтейнеров AVI был разработан Microsoft в те добрые времена, когда сама способность компьютера воспроизводить видео была еще в диковинку. В AVI нет документированной поддержки многих функций, которые есть в более новых форматах контейнеров, отсутствуют какие-либо метаданные и не поддерживается (по крайней мере официально) большинство используемых в наши дни видео- и аудиокодеков. Впоследствии разные фирмы реализовывали для формата AVI поддержку того или иного кодека (как правило, доработанный вариант был несовместим с прочими кодеками). По сей день для таких популярных кодировщиков,

как MEncoder (http://www.mplayerhq.hu/DOCS/HTML/en/encoding-guide.html), по умолчанию используется AVI-контейнер.

Видеокодеки

«Смотреть видео» — значит воспринимать один видеопоток и один аудиопоток, связанные между собой. Но для этих потоков не предусмотрено двух отдельных файлов, а есть один «видеофайл», например, в формате AVI или MP4. Как вы узнали из предыдущего раздела, это всего лишь контейнеры, как формат ZIP, файлы которого способны содержать внутри множество разнообразных файлов. Медиаконтейнер определяет способ хранения видео- и аудиопотоков в едином файле.

В то время как пользователь «смотрит видео», его компьютер решает множество задач. Нужно сделать следующее.

1. Проинтерпретировать формат контейнера, найти доступные видео- и аудиодорожку, установить, как они хранятся в файле и, следовательно, в каком порядке нужно расшифровывать данные.
2. Декодируя видеопоток, представлять его в виде последовательности картинок на экране.
3. Декодируя аудиопоток, посылать необходимые звуки в динамики.

Видеокодек — это алгоритм кодировки видеопотока, то есть механизм решения задачи 2 из названных выше («кодек» — сокращение от слов «кодировщик» и «декодировщик»). Видеопроигрыватель *расшифровывает* видеопоток с помощью кодека, а затем отображает последовательность картинок (кадров) на экране. Большинство современных кодеков умеет разными способами снизить объем информации, необходимой для показа каждого следующего кадра. Есть, к примеру, такие, которые сохраняют не покадровую развертку (последовательность скриншотов), а лишь отличия каждого следующего кадра от предыдущего. Поскольку от кадра к кадру видеоданные, как правило, меняются незначительно, таким способом можно достичь высокой степени сжатия — серьезно уменьшить размер файла.

Существует кодировка видео *с потерями* и *без потерь*. Видео, закодированное без потерь, занимает слишком много места и не может пригодиться в Сети, поэтому я сосредоточу свое внимание на кодеках, сжимающих с потерями. Сжатие с потерями означает, что при кодировке часть информации безвозвратно теряется. Кодировка видеоданных, как и, например, копирование аудиокассеты, влечет за собой потерю сведений об исходном видео и ухудшение качества. И если на аудиокассете, представляющей собой копию копии, возникает фоновый шум, то много раз перекодированное видео может выглядеть «блочным» (разбитым на квадратики), особенно в сценах, где много движения (так может быть и при кодировании качественного оригинала, если выбран плохой кодек или неправильно подобраны параметры). Впрочем, кодеки, сжимающие с потерями, замечательно умеют уменьшать объем, а некоторые даже позволяют «обмануть» человеческий глаз, отчего блочность, вызванная потерей информации, воспринимается не так резко.

Существует множество видеокодеков, из которых целям нашего рассмотрения отвечают три: H.264, Theora и VP8.

H.264

H.264 (http://ru.wikipedia.org/wiki/H.264) — видеокодек, известный еще как MPEG-4 part 10, MPEG-4 AVC, или более полно — MPEG-4 Advanced Video Coding. Разработанный группой MPEG, кодек H.264 был стандартизован в 2003 году. Он призван обеспечить единый стандарт кодирования видео для маломощных устройств с медленным сетевым подключением (мобильных аппаратов); высокомощных устройств с быстрым сетевым подключением (современных десктопов) и для всех промежуточных типов компьютеров. С данной целью стандарт H.264 разбит на «профили», в каждом из которых определен свой набор настроек; чем он сложнее и богаче, тем меньше будет размер закодированного файла. Надо отметить, что более «высокие» профили используют больше функций и поэтому демонстрируют лучшее качество картинки, качественнее сжимают информацию, но вместе с тем требуют больше времени на кодирование и больше процессорной мощности на декодирование в реальном времени.

Чтобы очертить круг имеющихся профилей, скажу, что аппараты iPhone поддерживают только базовый (Baseline) профиль, цифровой ресивер AppleTV — базовый и основной (Main), тогда как проигрыватель Adobe Flash на PC умеет воспроизводить H.264-видео базового, основного и высокого (High) профилей. Сервис YouTube, находящийся в собственности Google, применяет кодек H.264 для кодирования высококачественного видео, которое затем воспроизводится с помощью Adobe Flash. Кроме того, на YouTube публикуется H.264-видео для мобильных устройств: Apple iPhone и аппаратов с операционной системой Google Android. Кодек H.264 — один из кодеков, разрешенных спецификацией Blu-ray. Диски Blu-ray, содержимое которых закодировано с помощью этого кодека, обычно используют высокий профиль.

PC-несовместимые устройства, умеющие воспроизводить H.264-видео (в том числе iPhone и бытовые проигрыватели Blu-ray), как правило, декодируют видеопоток на специальном чипе, потому что их основным процессорам далеко до той мощности, которая нужна для расшифровки H.264 в реальном времени. Аппаратное декодирование H.264 поддерживают и многие видеокарты компьютеров-десктопов. Для кодирования видео в H.264 предусмотрено несколько альтернатив, среди которых есть библиотека x264 с открытым исходным кодом. Стандарт H.264 защищен патентом; за выдачу лицензий на пользование им отвечает группа MPEG LA. Видеопоток H.264 может быть вложен в большинство популярных контейнеров (см. предыдущий раздел), в том числе MP4, который преимущественно используется в интернет-магазине Apple iTunes, и MKV, большинство пользователей которого — энтузиасты некоммерческого видео.

Theora

Разработкой кодека Theora (http://ru.wikipedia.org/wiki/Theora), потомка технологии VP3, занимался фонд Xiph.org. Theora — это кодек, свободный от отчислений. Пользование им не ограничено никакими патентами, кроме исходных патентов на VP3, лицензия на которые бесплатна. В 2004 году стандарт был заморожен, но проект Theora, в состав которого входят эталонные кодировщик и декодировщик

с открытым исходным кодом, выпустил версию 1.0 лишь в ноябре 2008 года, а версию 1.1 — в сентябре 2009 года.

Видео в формате Theora может быть вложено в любой контейнер; чаще всего его можно видеть в контейнере Ogg. Поддержкой Theora сразу после установки обладают все основные дистрибутивы Linux. Браузер Mozilla Firefox версии 3.5 имеет встроенную поддержку видео Theora в Ogg-контейнере (под «встроенной» здесь подразумевается поддержка на всех платформах без вспомогательных программ, специфических для этих платформ). После установки программы-декодера Xiph.org можно воспроизводить видео Theora на платформах Windows и Mac OS X.

VP8

VP8 (http://ru.wikipedia.org/wiki/VP8) — это еще один видеокодек от компании On2, разработавшей VP3 (наследником которого является Theora). По качеству он сопоставим с H.264 базового профиля, но обладает большим потенциалом дальнейшего развития.

В 2010 году компания Google приобрела On2, после чего была опубликована спецификация кодека VP8 и раскрыт код его эталонных реализаций — кодировщика и декодировщика. В ходе этих мероприятий были также «открыты», то есть освобождены от платного лицензирования, все патенты, ранее полученные On2 на VP8.



ПРИМЕЧАНИЕ

В случае с патентованной технологией ни на что большее надеяться нельзя, потому что аннулировать уже выданный патент невозможно. Чтобы сделать технологию «дружественной» к открытому исходному коду, надо ее опубликовать под бесплатной лицензией, и тогда все желающие смогут пользоваться ею, не отчисляя ничего в пользу держателя патента.

Итак, по состоянию на май 2010 года VP8 — *бесплатный современный кодек, не ограниченный никакими патентами*, кроме тех, пользование которыми благодаря Google уже стало безвозмездным.

Аудиокодеки

Если вы не собираетесь выкладывать в Сеть только фильмы, снятые до 1927 года, то стоит задуматься об аудиодорожке в составе видеофайла. *Аудиокодеки*, как и видеокодеки, — это алгоритмы кодирования, применяемые (в данном случае) к аудиопотокам. Естественно, имеются аудиокодеки, сжимающие данные *с потерями* и *без потерь*. При сжатии аудио без потерь файлы из-за своих больших размеров получаются непригодными для публикации в Сети, поэтому я снова сконцентрируюсь на кодировании с потерями.

Аудиокодеки, сжимающие с потерями, делятся на несколько категорий. В таких областях, как телефония, аудиоряду не сопутствует видеоряд; в связи с этим есть много аудиокодеков, оптимальным образом кодирующих речь. Например, ими невозможно закодировать музыку с компакт-диска: результат будет не благозвучнее,

чем пение четырехлетнего ребенка в мегафон. Зато они применяются в Asterisk PBX[1], потому что трафик стоит денег, а эти специализированные кодеки сжимают человеческую речь во много раз эффективнее, чем аудиокодеки общего назначения. Поддержка речевых кодеков как в браузерах, так и в сторонних приложениях не предусмотрена; из-за этого они мало распространены в Сети. Далее я сосредоточусь на *сжимающих с потерями аудиокодеках общего назначения*.

Как упоминалось в разделе «Видеокодеки» этой главы, при просмотре видео ваш компьютер одновременно выполняет несколько операций.

1. Интерпретирует формат контейнера.
2. Декодирует видеопоток.
3. Декодирует аудиопоток и посылает необходимые звуки в динамики.

Аудиокодек отвечает за решение задачи 3. Он расшифровывает аудиопоток и превращает его в цифровые сигналы, которые в колонках или наушниках превращаются в звук. Как и в видеокодеках, при кодировке аудио применяются разные способы снижения объема информации аудиопотока. Поскольку мы говорим о сжатии с потерями, то в цикле «запись → кодирование → декодирование → воспроизведение» часть данных теряется безвозвратно. Различные кодеки отбрасывают разные фрагменты информации, руководствуясь при этом одной задачей: «обмануть» уши слушателя, чтобы тот не заметил недостающего.

Аудиоданным присуща одна черта, которой лишено видео: они могут выполнять дробление на *каналы*. Мы посылаем звук в динамики. А сколько динамиков подключено к вашему домашнему компьютеру? Могу предположить, что их всего два: один слева и один справа. У моего десктопа три динамика: слева, справа и внизу (на полу). В системах «объемного звука» может быть шесть или даже более колонок, специальным образом расставленных по комнате. И вот к каждой из колонок подводится один из каналов оригинальной записи. Идея такова: когда человек сидит посреди комнаты, а вокруг него из разных точек звучат шесть самостоятельных звуковых каналов, мозг начинает синтезировать этот звук и у слушателя возникает ощущение присутствия. Неужели это работает?.. Многомиллиардный рынок подсказывает нам, что да.

Большинство аудиокодеков общего назначения поддерживает два звуковых канала. При записи звук дробится на левый и правый каналы. Во время кодирования оба канала сводятся в единый поток, а при декодировании вновь разделяются и их данные направляются к соответствующим динамикам. Есть аудиокодеки, поддерживающие больше двух каналов. В них каждый канал отслеживается особо, благодаря чему проигрыватель может правильно распределить каналы между колонками.

Кажется, я говорил, что существует множество видеокодеков? Забудьте это. Аудиокодеков *непостижимо много*. В Интернете, однако, широко используются только три из них, о которых вам и надо знать: MP3, AAC, Vorbis.

MPEG-1 Audio Layer 3

Кодек MPEG-1 Audio Layer 3 (http://ru.wikipedia.org/wiki/MP3) в обиходе известен как MP3. Если вам никогда не приходилось слышать об MP3, не знаю, как и быть.

[1] Бесплатная программа для интернет-телефонии. — *Примеч. перев.*

В любом супермаркете сейчас под именем MP3-проигрывателей продаются портативные проигрыватели... Ну да ладно.

MP3 поддерживает *двухканальный звук*. Доступен различный *битрейт*: 64 Кбит/с, 128 Кбит/с, 192 Кбит/с и множество других (от 32 до 320). Чем выше битрейт, тем больше размер файла и тем лучше качество аудио, хотя отношение качества звука к битрейту не линейно (при битрейте 128 Кбит/с звук более чем вдвое качественнее, нежели при 64 Кбит/с, но 256 Кбит/с дает менее чем двойной прирост качества по сравнению со 128 Кбит/с). Кроме того, формат MP3, стандартизованный в 1991 году, допускает *кодирование с переменным битрейтом*, при котором одни части закодированного потока сжаты сильнее других. Так, паузу между звуками можно кодировать с очень низким битрейтом, а после паузы, когда на нескольких инструментах будет взят сложный аккорд, битрейт подскочит. Разрешается и *кодирование MP3 с постоянным битрейтом*.

Стандарт MP3 не определяет точный способ кодирования в MP3-формат (хотя определяет точный способ декодирования), поэтому разные реализации кодировщика пользуются различными психоакустическими моделями. Это создает огромный разброс результатов, но одни и те же проигрыватели умеют воспроизводить все полученные файлы. Лучший бесплатный MP3-кодировщик (а при среднем и высоком битрейте, возможно, лучший из всех) создан в открытом проекте LAME.

Формат MP3 защищен патентом, из-за чего «родного» декодирования MP3 в дистрибутивах Linux нет. Самостоятельные MP3-файлы воспроизводит в наши дни почти любой портативный проигрыватель, а MP3-аудиопоток может быть вложен в любой медиаконтейнер. Adobe Flash способен воспроизводить как самостоятельные MP3-файлы, так и MP3-аудиопотоки в контейнерах MP4.

Advanced Audio Coding

Кодек Advanced Audio Coding (http://ru.wikipedia.org/wiki/Advanced_Audio_Coding) известен под аббревиатурой AAC. Он был стандартизован в 1997 году и набрал силу после того, как компания Apple сделала его форматом по умолчанию в своем онлайн-магазине iTunes Store. Первоначально все AAC-файлы, приобретаемые пользователями iTunes Store, были зашифрованы проприетарной DRM-схемой Apple под названием FairPlay. Но теперь многие композиции в iTunes Store доступны в виде незащищенных AAC-файлов. Apple называет такой формат iTunes Plus, ведь назвать его iTunes Minus было бы как-то неудобно. *Формат AAC защищен патентом;* прейскурант на лицензирование можно найти в Интернете.

Задачей разработчиков AAC было добиться более высокого качества звука, чем в MP3 при таком же битрейте. В отличие от MP3, в котором определен набор доступных величин битрейта (верхняя граница — 320 Кбит/с), AAC допускает совершенно любой битрейт. Можно кодировать *до 48 каналов звука*, хотя это и не практикуется. Отличие формата AAC от MP3 еще и в том, что он, подобно H.264, имеет несколько профилей. AAC-аудио профиля «низкой сложности» может воспроизводиться в реальном времени на устройствах с ограниченной мощностью процессора. Более высокие профили обеспечивают при том же битрейте лучшее качество звука, но взамен требуют больше времени (или мощности) на кодирование-декодирование.

Все современные продукты Apple, в том числе iPod, AppleTV и QuickTime, поддерживают те или иные профили самостоятельного AAC-аудио и аудиопотоков в контейнерах MP4. Проигрыватель Adobe Flash и видеопроигрыватели с открытым исходным кодом Mplayer и VLC умеют работать со всеми профилями AAC в MP4-контейнерах. Для кодирования существует открытая библиотека FAAC, поддержку которой можно добавить (при компиляции) в программы mencoder и ffmpeg.

Vorbis

Кодек Vorbis (http://ru.wikipedia.org/wiki/Vorbis) часто называют Ogg Vorbis, что технически неверно, так как Ogg — всего лишь формат контейнера (см. раздел «Видеоконтейнеры» этой главы), и, кроме того, аудиопоток Vorbis может быть вложен в другие контейнеры. *Vorbis не патентован*, поэтому с ним умеют работать сразу после установки все основные дистрибутивы Linux, а также портативные устройства с открытой прошивкой Rock box. Браузер Mozilla Firefox 3.5 поддерживает воспроизведение Ogg-видеофайлов с аудиодорожками Vorbis. Мобильные устройства на платформе Android умеют воспроизводить самостоятельные аудиофайлы Vorbis. Обычно аудиопоток Vorbis вкладывается в контейнер Ogg или WebM, но также есть возможность вложить его в контейнер MP4, MKV или (с некоторыми дополнительными хитростями) AVI. Поддерживается *произвольное количество каналов звука*.

Существует несколько реализаций кодека Vorbis с открытым исходным кодом: кодировщики OggConvert, aoTuV, декодировщики ffmpeg и libvorbis. Имеются также компоненты QuickTime для Mac OS X и фильтры DirectShow для Windows.

Что работает в Интернете?

Если ваши глаза еще не устали от многочисленных подробностей, значит, дело идет хорошо. Вы, конечно, уже заметили, что видео и аудио — сложные для изучения темы. А ведь приведенные выше сведения — лишь минимум! Теперь вам, без сомнения, хочется знать, как все это относится к HTML5. Дело в том, что для встраивания видеофайлов на страницы в HTML5 предусмотрен тег `<video>`. Никаких ограничений на используемые видеокодеки, аудиокодеки и форматы контейнеров нет. Один тег `<video>` может ссылаться на несколько видеофайлов; из них браузер выберет самый первый, который сумеет воспроизвести. *Значит, ваша задача — узнать, в каких браузерах какие контейнеры и кодеки поддерживаются.*

В период написания этой книги расстановка сил в HTML5-видео была такой.

- Mozilla Firefox (3.5 и более новые версии) поддерживает видео Theora и аудио Vorbis в контейнере Ogg.
- Opera (10.5 и более новые версии) поддерживает видео Theora и аудио Vorbis в контейнере Ogg.
- Google Chrome (3.0 и более новые версии) поддерживает видео Theora и аудио Vorbis в контейнере Ogg, а также видео H.264 (всех профилей) и аудио AAC (всех профилей) в контейнере MP4.

- По состоянию на 9 июня 2010 года версии Google Chrome для разработчиков, ночные сборки Chromium, Mozilla Firefox и экспериментальные сборки Opera все без исключения поддерживают видео VP8 и аудио Vorbis в контейнере WebM[1].
- Safari на платформах Mac и Windows PC (3.0 и более новые версии) поддерживает все то же самое, что и QuickTime. Теоретически можно было бы предложить пользователям установить сторонние приложения QuickTime, но на практике вряд ли кто-то будет утруждаться этим. Остается пользоваться форматами, поддержка которых есть в QuickTime по умолчанию. В их длинный список не входят видео Theora, аудио Vorbis и формат контейнеров Ogg. В QuickTime, однако, *есть* поддержка видео H.264 (основного профиля) и аудио AAC в контейнере MP4.
- Такие мобильные устройства, как iPhone и смартфоны на платформе Google Android, поддерживают видео H.264 (базового профиля) и аудио AAC (профиля низкой сложности) в контейнере MP4.
- Adobe Flash (9.0.60.184 и более новые версии) поддерживает видео H.264 (всех профилей) и аудио AAC (всех профилей) в контейнере MP4.
- Internet Explorer 9, как ожидается, будет поддерживать какие-то из профилей видео H.264 и аудио AAC в контейнере MP4.
- Internet Explorer 8 вообще не поддерживает HTML5-видео, но почти у всех его пользователей установлен Adobe Flash. Далее в этой главе я покажу, как использовать Flash в качестве резервного варианта при HTML5-видео.

В табл. 5.2 содержится та же самая информация в более удобном для восприятия виде.

Таблица 5.2. Поддержка видеокодеков в стабильных браузерах

Кодек/контейнер	IE	Firefox	Safari	Chrome	Opera	iPhone	Android
Theora+Vorbis+Ogg	–	3.5+	–	5.0+	10.5+	–	–
H.264+AAC+MP4	–	–	3.0+	5.0+	–	3.0+	2.0+
WebM	–	–	–	–	–	–	–

Спустя год расстановка сил существенно поменяется: поддержка WebM будет реализована в стабильных версиях многих браузеров и пользователи начнут переходить на эти стабильные версии. Ожидается, что поддержка кодеков будет такой, как показано в табл. 5.3.

Таблица 5.3. Поддержка видеокодеков в ожидаемых версиях браузеров

Кодек/контейнер	IE	Firefox	Safari	Chrome	Opera	iPhone	Android
Theora+Vorbis+Ogg	–	3.5+	–	5.0+	10.5+	–	–
H.264+AAC+MP4	–	–	3.0+	5.0+	–	3.0+	2.0+
WebM	9.0+*	4.0+	–	6.0+	11.0+	–	**

* Internet Explorer 9 будет поддерживать WebM «только в том случае, если пользователь установил кодек VP8»; тем самым Microsoft заявляет, что в дистрибутив IE9 этот кодек включен не будет.

** Google обещал добавить поддержку WebM в систему Android в следующем релизе, дата которого еще не определена.

[1] За более свежей информацией обращайтесь на webmproject.org, где доступны ссылки на установочные файлы браузеров, совместимых с WebM. — *Примеч. авт.*

Теперь приготовьтесь к сокрушительному удару...

СЕКРЕТЫ РАЗМЕТКИ

Такого сочетания медиаконтейнера и кодеков, которое бы работало во всех браузерах с поддержкой HTML5, не существует.
По-видимому, в ближайшем будущем ситуация не изменится.

Чтобы ваше видео можно было просматривать на любом устройстве с любой рабочей средой, его придется кодировать несколько раз.

Если вы стремитесь добиться наилучшей совместимости с существующими платформами, действуйте так.

1. Создайте одну версию: видео в формате Theora и аудио Vorbis в Ogg-контейнере.
2. Создайте другую версию: WebM-контейнер, кодеки VP8 и Vorbis.
3. Создайте третью версию: видео H.264 базового профиля, аудио AAC профиля низкой сложности в контейнере MP4.
4. Сошлитесь на эти три видеофайла из одного тега `<video>`. Как запасной вариант подготовьте видеопроигрыватель на основе Flash.

Проблемы лицензирования видео H.264

Прежде чем продолжить, я не могу не отметить, что за кодирование видеоданных приходится платить двойную цену. Речь идет не только об очевидно дублирующемся процессе кодировки одного и того же файла в разные форматы, что требует лишних затрат машинного времени. Я говорю о вполне осязаемой цене: лицензионных отчислениях за пользование кодеком H.264.

Когда я впервые рассказывал о видеоформате H.264 (см. подраздел «H.264» раздела «Видеокодеки» этой главы), я упомянул, что этот видеокодек защищен патентом и лицензии на его использование выдает Консорциум MPEG LA. Это, как оказывается, очень важно, а почему важно — разъяснит статья H.264 Licensing Labyrinth, написанная Тимом Сиглином (Tim Siglin)[1]:

> Лицензионное портфолио H.264 разделено MPEG LA на две сублицензии: одна — для производителей программ-кодировщиков, а другая — для лиц, распространяющих видеоданные. [...]
> Сублицензия на вещание далее подразделяется на четыре ключевые категории, две из которых (подписка и одноразовая покупка, называемая также платным использованием) относятся к ситуации, когда конечный пользователь непосредственно платит за видеосервисы; другие две («бесплатное телевидение» и интернет-вещание) предусматривают платежи из иных источников, нежели средства конечного пользователя. [...]
> Лицензионные отчисления за «бесплатное телевидение» бытуют в двух вариантах. Первый представляет собой одноразовый платеж размером $2500 за кодировщик AVC-передачи, то есть ровно один AVC-кодировщик, «используемый получателем лицензии или по его доверенности для передачи AVC-видео конечному пользователю». Пользователь, в свою очередь, будет декодировать и просматривать видео. Если вы спросите, не двойной ли получается оплата, то ответ будет положительным, ведь лицензионный сбор уже уплачен производителем программы-кодировщика, а вещание также обложено отчислениями (одного из двух типов).

[1] http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/The-H.264-Licensing-Labyrinth-65403.aspx. — *Примеч. авт.*

> Второй вариант лицензионного сбора — годовая лицензия на вещание. [...] В этом случае сбор зависит от размера обслуживаемой аудитории:
> - $2500 за календарный год для сетей вещания на 100 000–499 999 домохозяйств;
> - $5000 за календарный год для сетей вещания на 500 000–999 999 домохозяйств;
> - $10 000 за календарный год для сетей вещания на 1 000 000 и более домохозяйств.
>
> [...] Однако при всех сложностях с «бесплатным телевидением», как это затрагивает интересы лиц, вовлеченных в распространение видео помимо сетей вещания? Как сказано выше, оплачиваться должно любое распространение контента. Оговорив, что «бесплатное телевидение» не ограничивается [обычным телевещанием], Консорциум MPEG LA установил сборы за интернет-вещание, понимаемое как «распространение AVC-видео по Всемирной сети конечным пользователям таким образом, что они не платят за право получать и просматривать видео». Иными словами, всякое вещание — [обычное телевизионное,] кабельное, спутниковое или посредством Интернета — облагается лицензионным сбором. [...]
>
> Компенсация за интернет-вещание обещает быть выше, чем за другие формы, так как аудитория Интернета растет гораздо быстрее, чем аудитория обычного телевидения, кабельного и спутникового «бесплатного телевидения». Таким образом, к сборам с вещательного рынка «бесплатного телевидения» прибавится еще один сбор, но MPEG LA предоставляет отсрочку до конца первого года действия лицензии, то есть 31 декабря 2010 года. После этого, как отмечает Консорциум, «отчисления будут не выше экономического эквивалента отчислений, которые можно было бы получить в то же время с сопоставимых сетей бесплатного телевидения».

В последний пункт, касающийся сборов с интернет-вещания, уже внесена поправка. Недавно MPEG LA объявил, что бесплатный обмен видеоданными в Интернете продлен до 31 декабря 2015 года. Одному Богу известно, что будет после этого.

Кодирование Ogg-видео с помощью Firefogg[1]

Firefogg — расширение Firefox с открытым исходным кодом, распространяемое под лицензией GPL. Оно позволяет кодировать Ogg-видео. Для использования Firefogg надо установить Mozilla Firefox версии 3.5 или более свежей, а затем зайти на сайт проекта. Главная страница этого сайта показана на рис. 5.1.

Щелкните на ссылке Install Firefogg (Установить Firefogg). Браузер спросит, действительно ли вы хотите установить расширение. Нажмите кнопку Allow (Разрешить) для продолжения установки (рис. 5.2).

Появится стандартное окно установки программ в Firefox. Чтобы продолжить, нажмите кнопку Install Now (Установить сейчас) (рис. 5.3).

Для завершения установки нажмите кнопку Restart Firefox (Перезапустить Firefox) (рис. 5.4).

Когда браузер перезапустится, сайт Firefogg подтвердит, что установка прошла успешно (рис. 5.5).

Чтобы запустить процесс кодирования, щелкните на ссылке Make web video (Создать веб-видео) (рис. 5.6), затем выберите исходный файл, нажав кнопку Select file (Выбрать файл) (рис. 5.7).

---

[1] Здесь термином «Ogg-видео» кратко обозначается видео Theora и аудио Vorbis в контейнере Ogg. Mozilla Firefox и Google Chrome имеют встроенную поддержку этого сочетания кодеков с контейнером. — *Примеч. авт.*



Рис. 5.1. Домашняя страница Firefogg



Рис. 5.2. Разрешим установку Firefogg

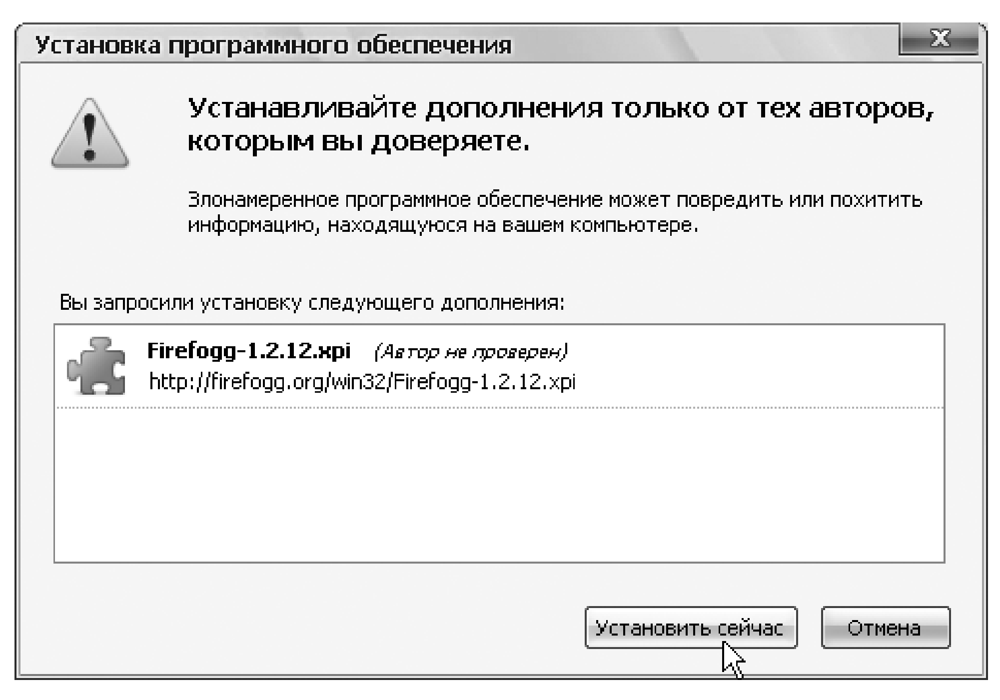


Рис. 5.3. Установка Firefogg

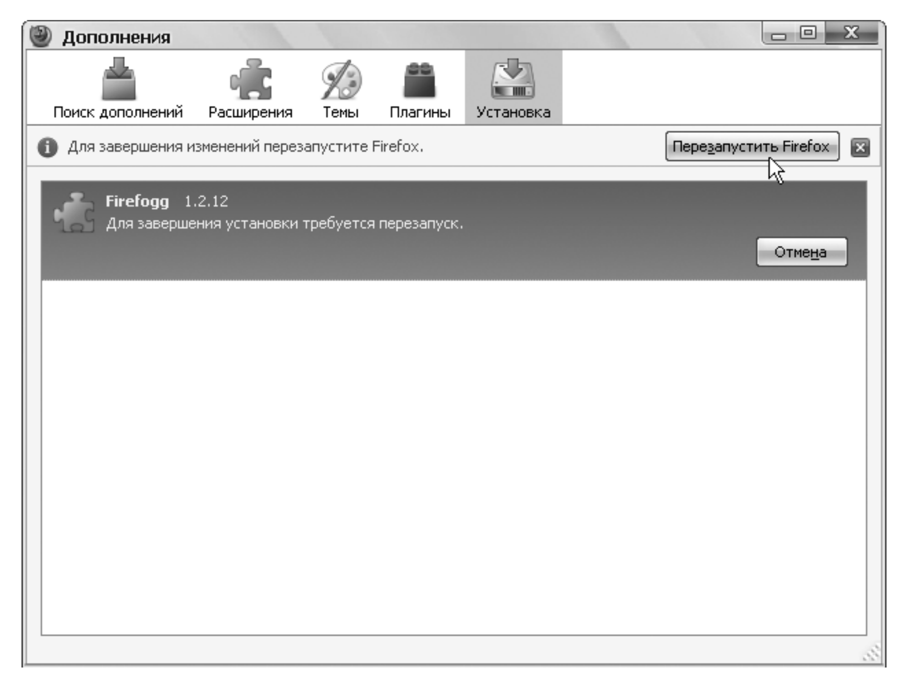


Рис. 5.4. Повторно запустим Firefox



Рис. 5.5. Установка прошла успешно



Рис. 5.6. А не закодировать ли нам видео?

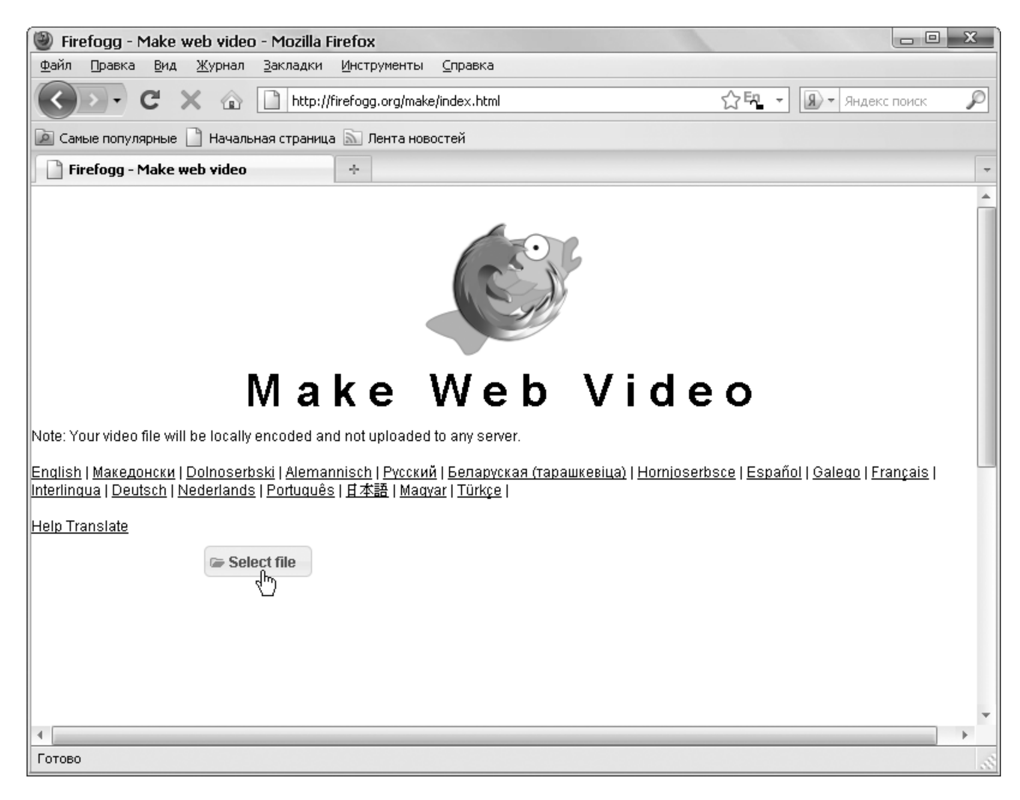


Рис. 5.7. Выберем исходный видеофайл

Основной интерфейс Firefogg состоит из шести вкладок (рис. 5.8).

- Presets (Шаблоны) — по умолчанию здесь выбрано Web-video (Веб-видео), что вполне отвечает нашим целям.
- Encoding range (Кодируемый интервал). Кодирование видео — небыстрый процесс. При знакомстве с программой стоит попробовать закодировать часть исходного видеофайла (например, первые 30 секунд), чтобы затем экспериментировать с настройками.
- Basic quality and resolution control (Базовые настройки качества и разрешения) — здесь находится большинство настроек, важных для нас.
- Metadata (Метаданные) — не буду писать об этом подробно, но вам следует знать, что закодированное видео можно оснащать метаданными: название, автор и т. п. Вы, вероятно, хоть раз добавляли к своим музыкальным коллекциям метаданные с помощью iTunes или другой программы-менеджера; здесь происходит то же самое.
- Advanced video encoding controls (Дополнительные настройки видеокодирования) — не меняйте здесь ничего, если не владеете нужными знаниями. О большинстве представленных здесь настроек пишет интерактивная справка Firefogg. Чтобы вызвать ее, щелкните на букве i рядом с интересующей настройкой.

- Advanced audio encoding controls (Дополнительные настройки аудиокодирования) — здесь тоже лучше ничего не менять.

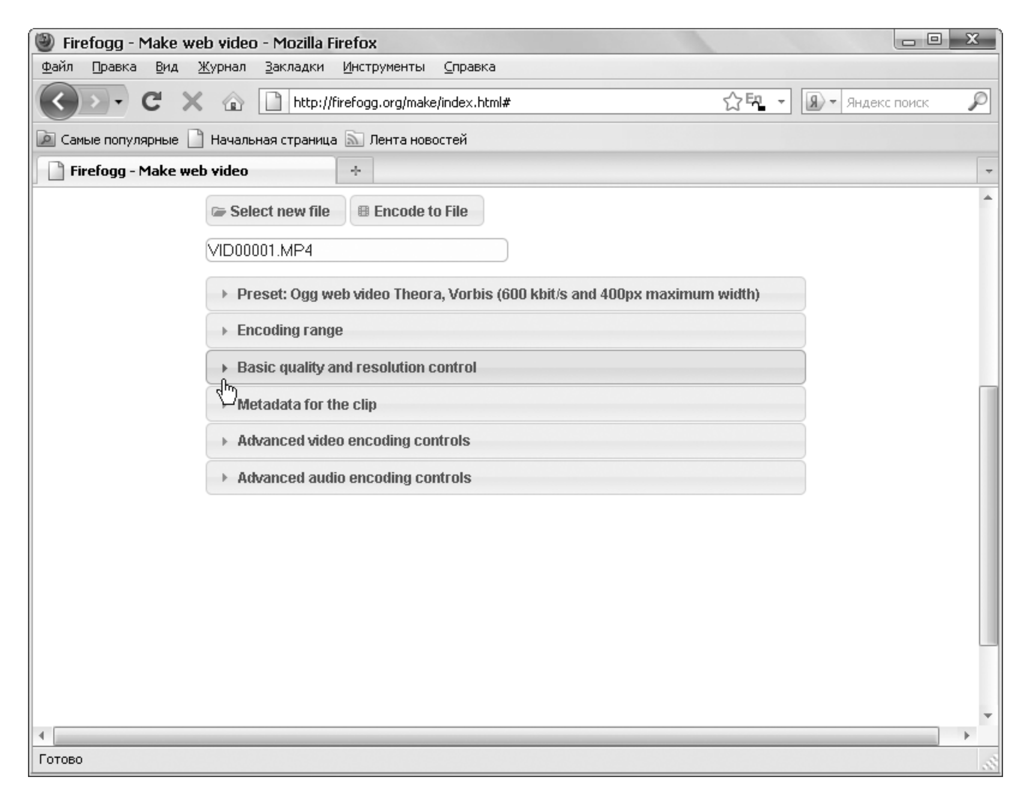


Рис. 5.8. Основной интерфейс Firefogg

Подробно рассмотрим только вкладку Basic quality and resolution control (Базовые настройки качества и разрешения) (рис. 5.9), где содержатся все важнейшие настройки.

- Video quality (Качество видео) — шкала размечена от 0 (наихудшее качество) до 10 (наилучшее качество). Чем больше выбранное число, тем больше по объему получится файл. Чтобы найти оптимальное для ваших требований соотношение объема и качества, надо экспериментировать.
- Audio quality (Качество аудио) — шкала размечена от 0 (наихудшее качество) до 10 (наилучшее качество). Как и в случае с видео, чем больше выбранное число, тем больше по объему получится файл.
- Video codec (Видеокодек) — здесь должно быть выбрано Theora.
- Audio codec (Аудиокодек) — здесь должно быть выбрано Vorbis.
- Video width (Ширина видео) и Video height (Высота видео) — по умолчанию значения в этих полях приравниваются к ширине и высоте кадров исходного файла. Если хотите в процессе кодирования поменять горизонтальный или вертикальный

размер, впишите желаемое значение здесь. Firefogg автоматически определит значение другой координаты, при котором сохраняется исходное соотношение сторон кадра, так что картинка вашего видеофайла не будет сжатой или растянутой.

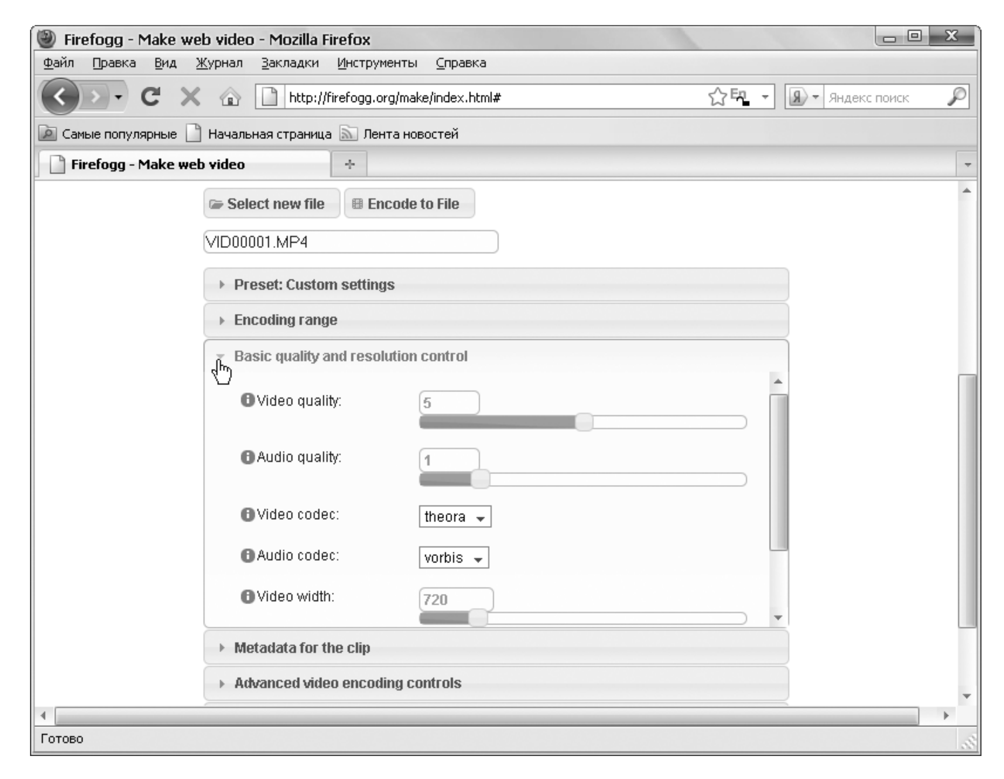


Рис. 5.9. Базовые настройки качества и разрешения

На рис. 5.10 показано, как я вдвое уменьшил ширину исходного видео. Заметьте, что Firefogg автоматически подбирает нужную высоту.

После выставления всех настроек нажмите кнопку Encode to File (Закодировать в файл), чтобы запустить процесс кодирования (рис. 5.11). При этом Firefogg попросит ввести имя для закодированного файла.

Во время кодирования Firefogg будет показывать красивый индикатор (рис. 5.12). Остается только ждать.

Пакетное кодирование Ogg-видео с помощью ffmpeg2theora

Существует несколько офлайновых программ-кодировщиков Ogg-видео. Если вам надо закодировать большое количество файлов и вы желаете автоматизировать процесс, обязательно воспользуйтесь программой ffmpeg2theora (http://v2v.cc/~j/ffmpeg2theora/).

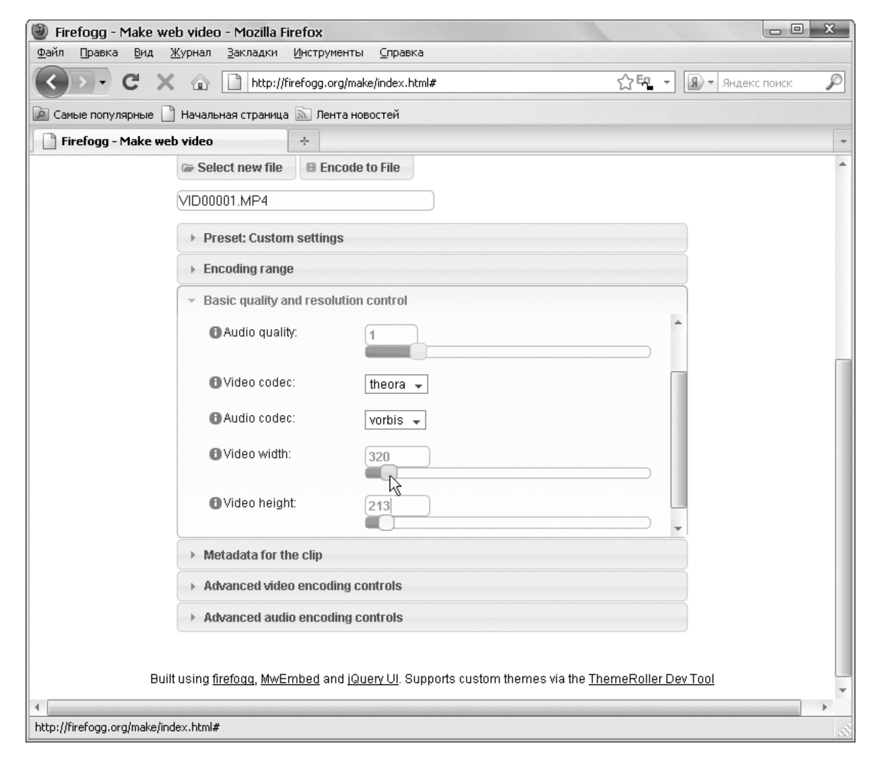


Рис. 5.10. Выбор ширины и высоты кадра

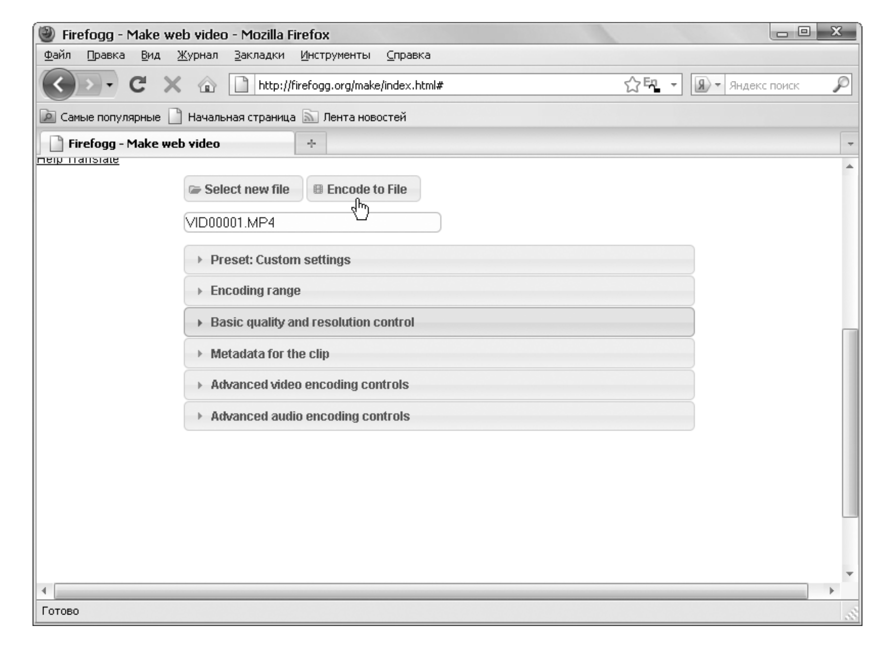


Рис. 5.11. Запустим процесс кодирования

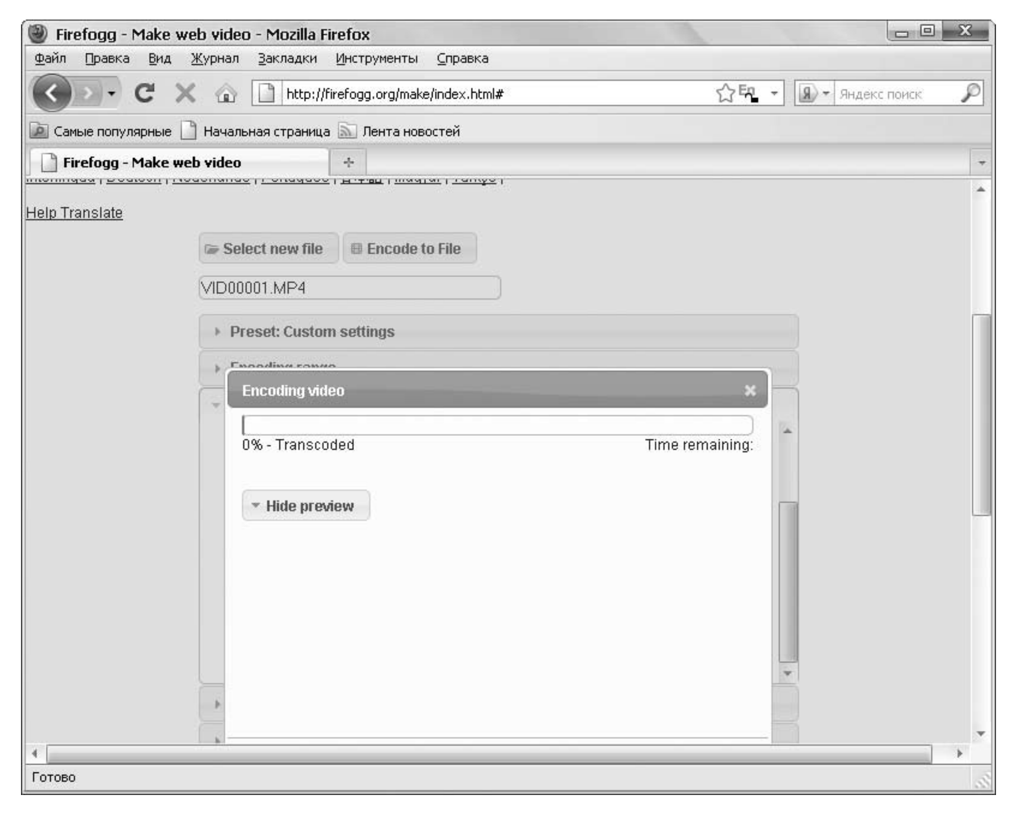


Рис. 5.12. Идет кодирование...

ffmpeg2theora — это программа для кодирования Ogg-видео с открытым исходным кодом, распространяемая под лицензией GPL. Доступны готовые сборки для Mac OS X, Windows и современных дистрибутивов Linux. В качестве исходного при кодировании может выступать практически любой видеофайл, в том числе цифровое видео (DV), записываемое камерами потребительского уровня.

Чтобы начать работу с ffmpeg2theora, вызовите ее из командной строки. В Windows это Главное меню ▸ Программы ▸ Стандартные ▸ Командная строка. В Mac OS X — Приложения ▸ Утилиты ▸ Терминал.

Программа ffmpeg2theora понимает много разных аргументов командной строки (флагов). Подробнее о них можно прочитать, вызвав `ffmpeg2theora --help`, я же сосредоточусь на трех аргументах.

- `--video-quality` *`Q`*, где *`Q`* принимает числовые значения от 0 до 10.
- `--audio-quality` *`Q`*, где *`Q`* принимает числовые значения от –2 до 10.
- `--max_size` *`W × H`*, где *`W`* и *`H`* — это соответственно максимальная ширина и высота кадра в видеофайлах, которые вы хотите получить на выходе (заметьте, что между первым и вторым числами должна быть набрана латинская буква x, а не какой-либо другой символ). Программа ffmpeg2theora подберет размер, сохраняющий пропорции сторон, так что размеры кадра в выходном файле могут быть меньше,

чем *W* × *H*. Так, если при кодировании видео размерами 720 × 480 пикселов назначить --max_size 320 × 240, то получится видеофайл с кадрами 320 пикселов шириной и 213 пикселов высотой.

Вот каким образом можно закодировать видео с теми же настройками, что и в предыдущем разделе:

```
you@localhost$ ffmpeg2theora --videoquality 5
                             --audioquality 1
                             --max_size 320x240
                             pr6.dv
```

Закодированный файл будет сохранен с расширением OGV в той же папке, что и исходное видео. Чтобы изменить это умолчание, надо передать ffmpeg2theora в командной строке аргумент вида --output=/*путь_к_закодированному_файлу*.

Кодирование H.264-видео с помощью HandBrake[1]

Если оставить в покое вопросы лицензирования (см. раздел «Проблемы лицензирования видео H.264» этой главы), то проще всего создавать H.264-видео с помощью программы HandBrake (http://handbrake.fr). HandBrake — это программа для кодирования H.264-видео с открытым исходным кодом, распространяемая под лицензией GPL (раньше она знала и другие видеоформаты, но в последней версии разработчики прекратили поддержку большинства из них, чтобы сосредоточить все свои усилия на H.264-видео). Доступны готовые сборки для Mac OS X, Windows и современных дистрибутивов Linux (http://handbrake.fr/downloads.php).

HandBrake существует в двух формах: как приложение с графическим интерфейсом и утилита командной строки. Сначала я покажу вам графический интерфейс, а потом мы выясним, как найденные нами оптимальные установки превратить в последовательность команд терминала.

Открыв программу HandBrake, выберите исходный видеофайл (рис. 5.13). Щелкните на пункте Source (Источник) и, выбрав Video File (Видеофайл), найдите нужный файл. HandBrake может принимать в качестве исходного практически любой файл, в том числе цифровое видео (DV), записываемое камерами потребительского уровня.

HandBrake предупредит, что вы не выбрали папку, в которой бы по умолчанию сохранялись закодированные файлы (рис. 5.14). Можно оставить без внимания это предупреждение, а можно открыть окно настроек (пункт меню Tools (Инструменты)) и назначить директорию по умолчанию для результатов работы программы.

---

[1] Здесь термином «H.264 video» кратко обозначается видео H.264 базового профиля и аудио AAC профиля низкой сложности в контейнере MPEG-4. Safari, Adobe Flash, а также iPhone и устройства на платформе Google Android имеют встроенную поддержку этого сочетания кодеков с контейнером. — *Примеч. авт.*

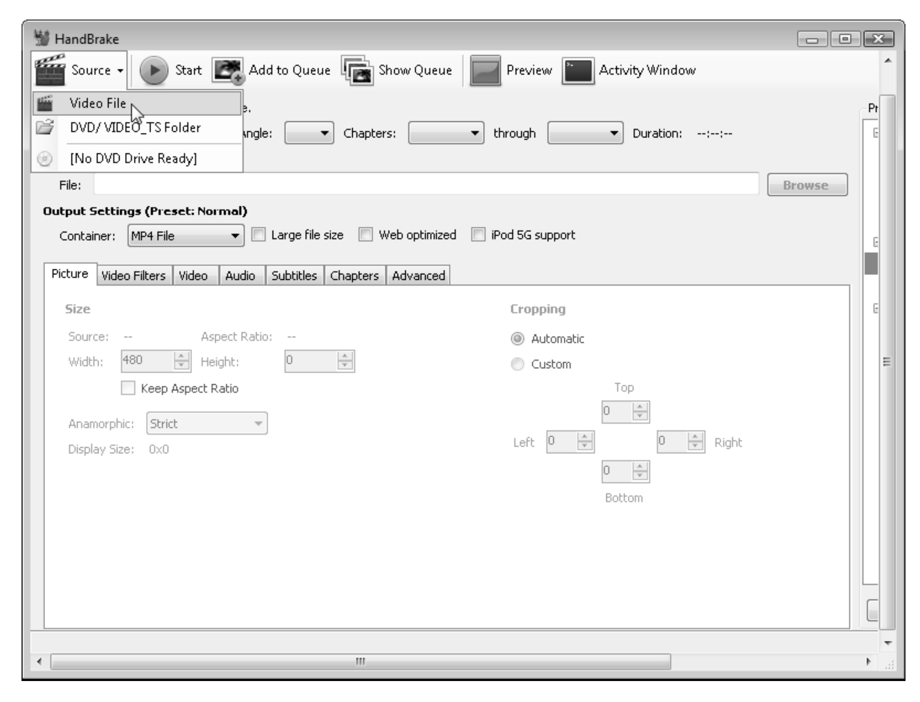


Рис. 5.13. Выберем исходный видеофайл

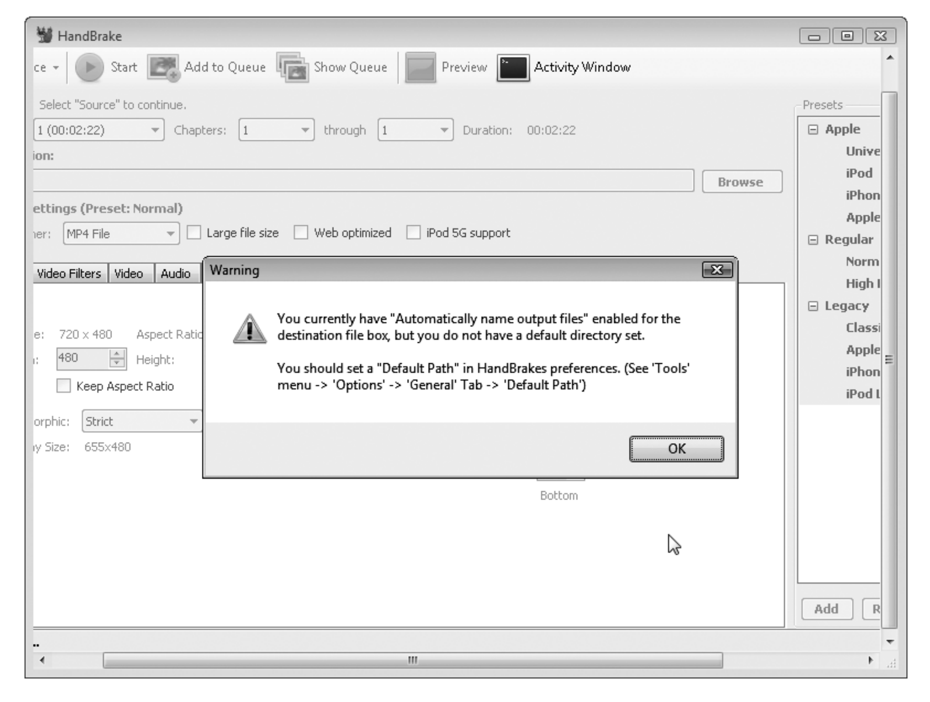


Рис. 5.14. Проигнорируем предупреждение

В правой части окна имеется список шаблонов. Если выбрать шаблон iPhone & iPod Touch, как показано на рис. 5.15, то большинство нужных нам настроек будут выставлены автоматически. По умолчанию отключена важная настройка веб-оптимизации (флажок Web optimized). Если установить флажок, как показано на рис. 5.16, то метаданные файла будут организованы так, что пользователь сможет приступить к просмотру, не дожидаясь конца загрузки, которая будет тем временем продолжаться в фоновом режиме. Рекомендую всегда устанавливать этот флажок. На качество картинки и размер закодированного файла эта настройка не влияет, так что нет причин избегать ее.

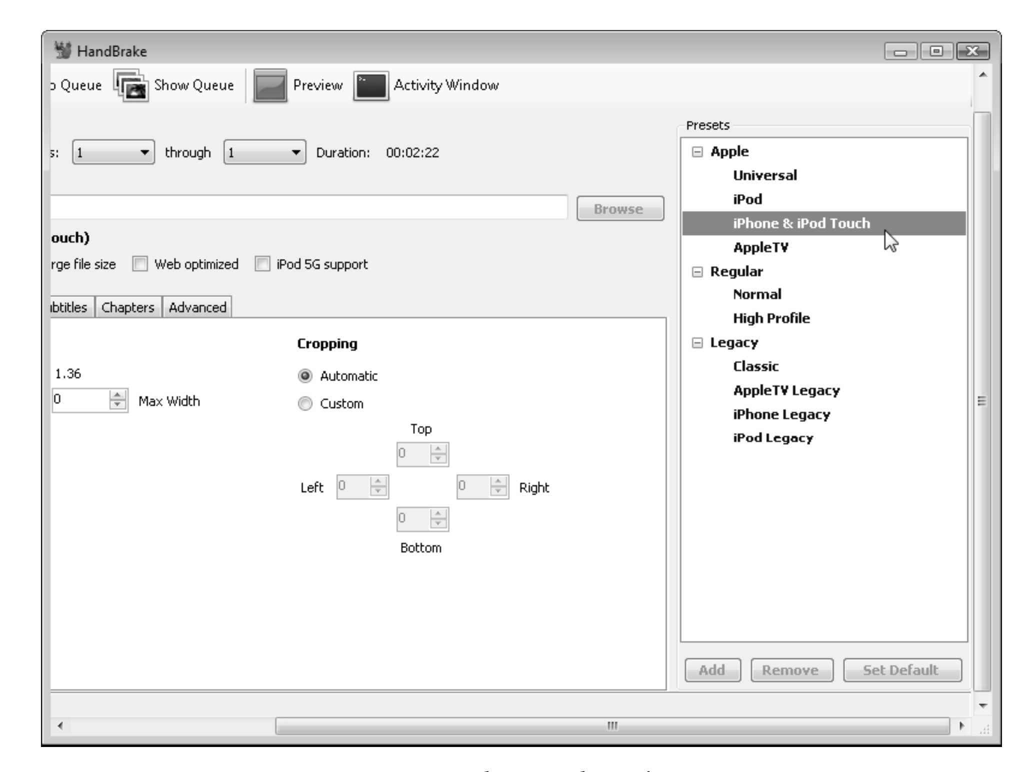


Рис. 5.15. Выберем шаблон iPhone

На вкладке Picture (Картинка) (рис. 5.17) можно выбрать максимальную высоту и ширину кадров закодированного видеофайла. Чтобы программа HandBrake не растягивала и не сжимала кадры при обработке, следует выставить флажок Keep Aspect Ratio (Сохранить пропорции).

На вкладке Video (Видео), изображенной на рис. 5.18, представлено несколько важных настроек.

- Video Codec (Видеокодек) — убедитесь, что выбрано значение H.264 (x264).
- 2-Pass Encoding (Кодирование в два прохода) — если установлен этот флажок, то HandBrake запустит кодирующий модуль дважды. В первый раз программа анализирует исходный видеофайл: сочетания цветов в нем, движение на экране,

разбивку на сцены. Второй проход — собственно кодирование, привлекающее информацию, которая была получена во время первого прохода. Как и следует ожидать, кодирование в два прохода будет примерно вдвое дольше по времени, зато на выходе получится файл лучшего качества без прироста в объеме. Я всегда выполняю кодирование H.264-видео в два прохода и советую делать вам то же самое, если, конечно, вы не строите второй YouTube и вам не приходится возиться с кодированием видео 24 часа в сутки.

- Turbo first Pass (Первый проход в турборежиме) — при двухпроходной кодировке благодаря этой настройке можно сэкономить часть времени. Она уменьшает объем работы, выполняемой во время первого прохода кодировщика, что, однако, лишь немного снижает качество. Я обычно устанавливаю этот флажок, но если вы цените качество превыше всего, снимите флажок.
- Quality (Качество) — качество кодируемого видео определяется несколькими способами. Можно указать желаемый объем конечного файла, и тогда HandBrake сделает все возможное, чтобы вместить итог своих трудов в заявленный объем. Как альтернативу можно указать средний битрейт, то есть буквально количество бит, приходящихся на одну секунду кодированного видео (о «среднем» битрейте говорят потому, что не все секунды требуют одинакового количества бит). Можно, наконец, выбрать постоянный коэффициент качества на шкале от 0 до 100 %. Чем выше процент, тем лучше качество и тем больше файл. Нельзя однозначно сказать, какой из настроек качества лучше пользоваться.

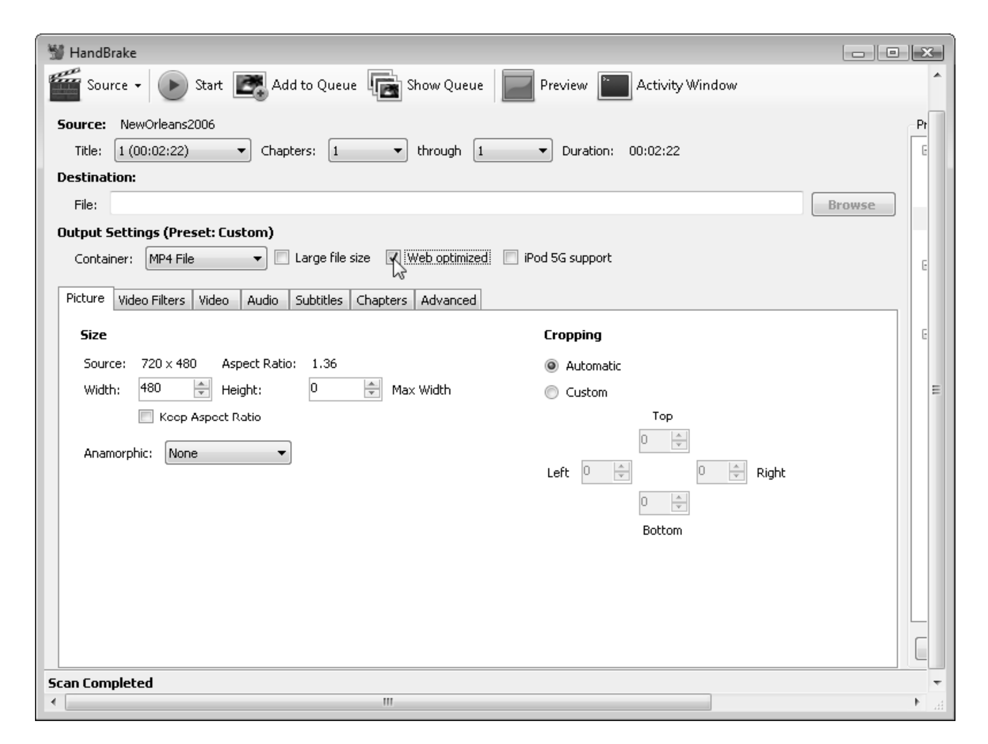


Рис. 5.16. Не забудем о веб-оптимизации

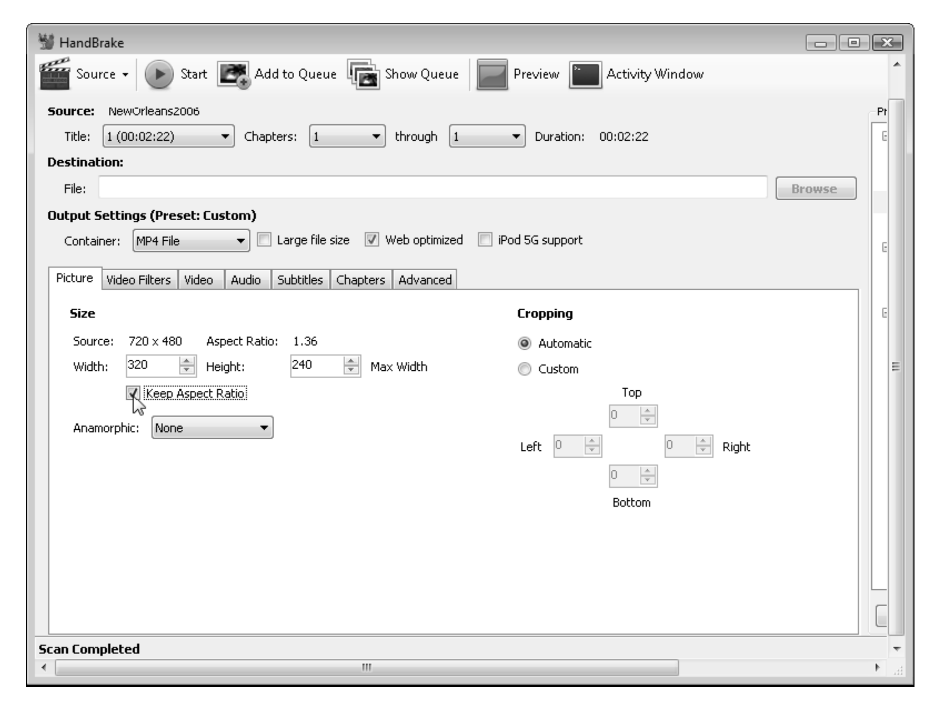


Рис. 5.17. Назначим горизонтальный и вертикальный размеры

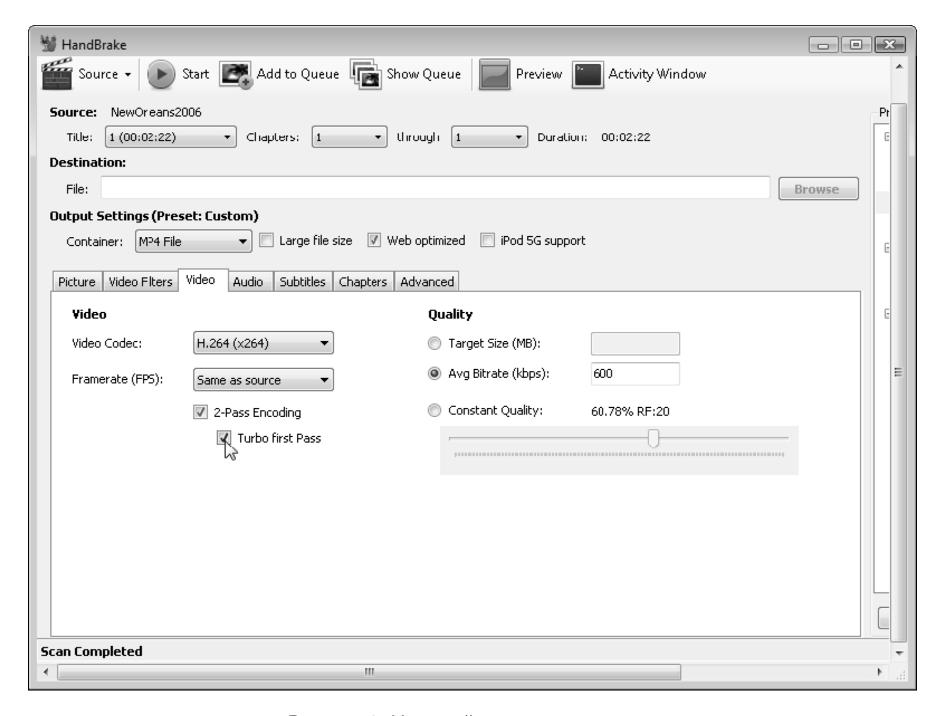


Рис. 5.18. Настройки качества видео

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Можно ли кодировать Ogg-видео тоже в два прохода?

Ответ: Да, но ввиду фундаментальных отличий в алгоритме кодирования это вряд ли необходимо. Кодирование H.264-видео в два прохода почти всегда повышает качество картинки, тогда как двухпроходное кодирование Ogg-видео способно только снизить размер файла до определенной степени (возможно, вы заинтересованы именно в этом, но мой опыт свидетельствует о нецелесообразности такой меры, да и лишнего времени на кодирование веб-видео тратить не стоит). Наилучшее качество Ogg-видео обеспечивают настройки качества видео, а о кодировании в два прохода беспокоиться незачем.

В данном примере я установил средний битрейт 600 Кбит/с, что довольно много для видео с разрешением 320 × 240. Я разрешил кодирование в два прохода с первым проходом в турборежиме.

На вкладке Audio (Аудио), которую показывает рис. 5.19, вам вряд ли придется что-либо менять, кроме двух настроек. Во-первых, если исходный файл содержит несколько аудиодорожек, то, возможно, придется выбрать, какую из них оставить в закодированном файле. Во-вторых, если звуковое сопровождение вашего видео — это преимущественно речь людей, а не музыка и звуки окружающего мира, то битрейт аудиодорожки можно понизить до 96 Кбит/с или около того. Прочие настройки, унаследованные из шаблона iPhone, оптимальны.

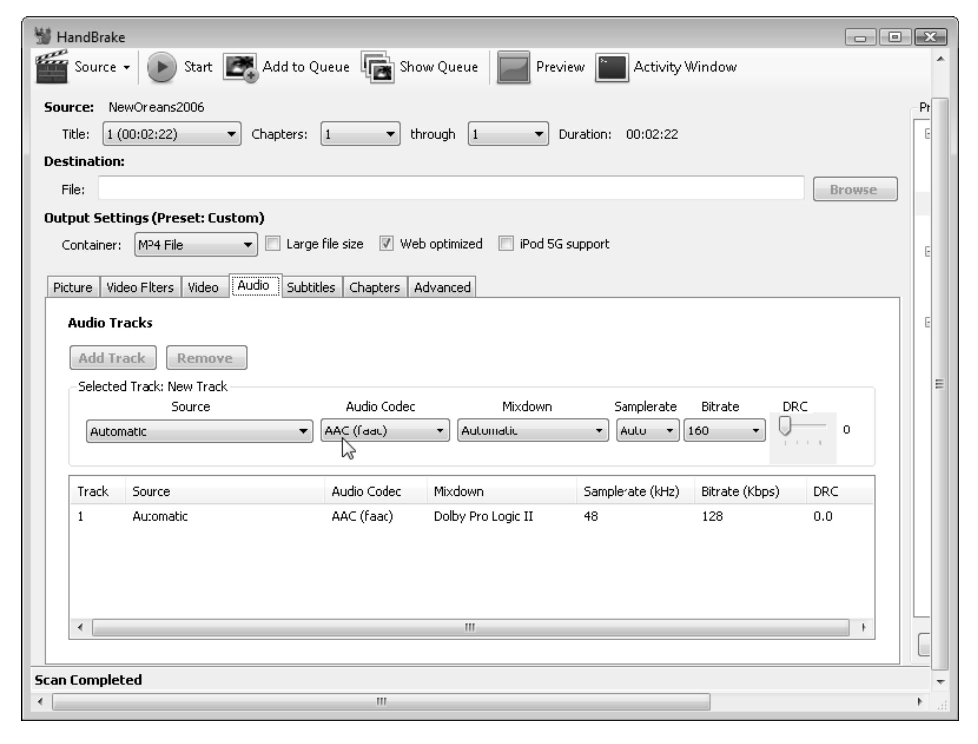


Рис. 5.19. Настройки качества аудио

Нажмите кнопку Browse (Обзор) (рис. 5.20), чтобы выбрать имя закодированного видеофайла и место его сохранения.

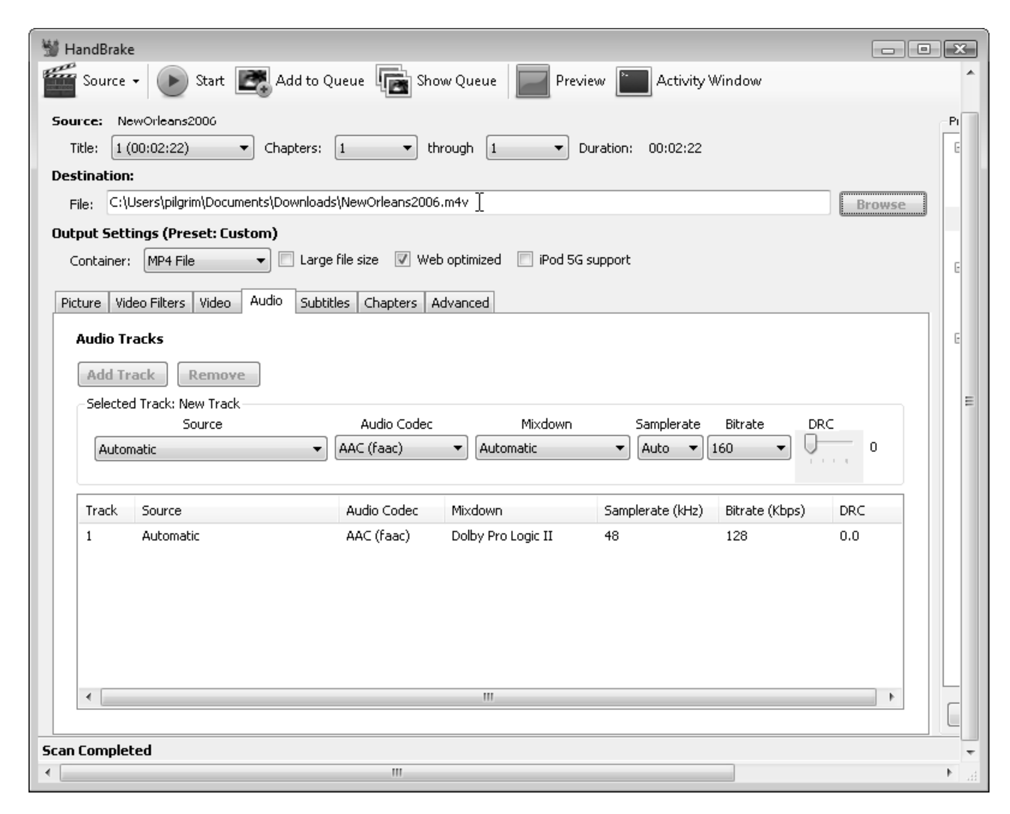


Рис. 5.20. Выберем имя конечного файла

Чтобы приступить к кодированию, щелкните на пункте Start (Запуск) (рис. 5.21).

По мере преобразования вашего видеофайла программа HandBrake будет выводить кое-какую статистику (рис. 5.22).

Пакетное кодирование H.264-видео с помощью HandBrake

Выше было упомянуто о варианте HandBrake для командной строки. Свежую версию этого «издания» можно скачать по адресу http://handbrake.fr/downloads2.php.

Программа HandBrake для командной строки, подобно ffmpeg2theora (см. раздел «Пакетное кодирование Ogg-видео с помощью ffmpeg2theora» этой главы), может принимать множество разных аргументов. Подробнее о них можно прочесть, набрав `HandBrakeCLI --help` в окне терминала или командной строке, а я расскажу лишь о нескольких.

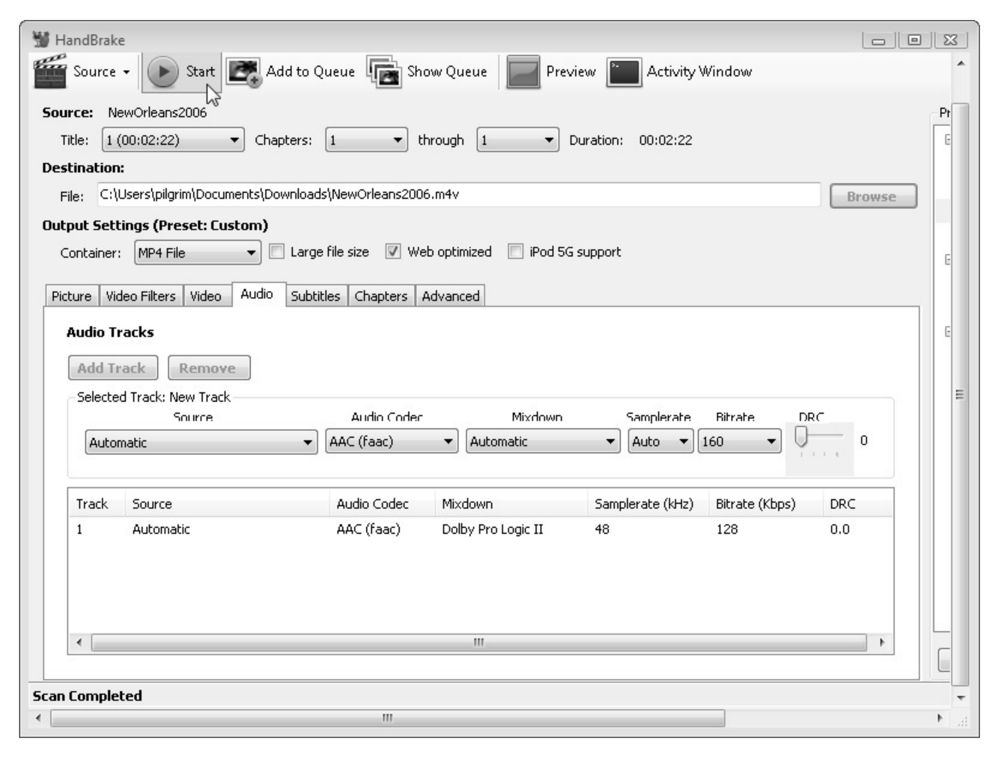


Рис. 5.21. Итак, не закодировать ли нам видео?

```
C:\Windows\System32\cmd.exe
Encoding: task 1 of 2, 57.75 % (126.81 fps, avg 118.71 fps, ETA 00h00m15s)
```

Рис. 5.22. Спокойствие, только спокойствие...

- `--preset "X"`, где *X* — название шаблона HandBrake (оно обязательно должно быть заключено в кавычки). Шаблон для сетевого H.264-видео называется iPhone & iPod Touch.
- `--width W`, где *W* — ширина кадров закодированного видеофайла. HandBrake автоматически подберет высоту так, чтобы исходные пропорции были сохранены.
- `--vb Q`, где *Q* — средний битрейт (Кбит/с).
- `--two-pass` — флаг, включающий кодирование в два прохода.
- `--turbo` — флаг, разрешающий при двухпроходной кодировке первый проход в турборежиме.
- `--input F`, где *F* — имя исходного видеофайла.
- `--output E`, где *E* — имя конечного (закодированного) видеофайла.

Рассмотрим, как можно, работая в командной строке, передать HandBrake те же самые настройки кодирования, которые мы ранее выставили в графической версии программы:

```
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch"
                            --width 320
                            --vb 600
                            --two-pass
                            --turbo
                            --input pr6.dv
                            --output pr6.mp4
```

Разберемся с этой записью по порядку. Пользователь запускает HandBrake с шаблоном iPhone & iPod Touch, устанавливает размеры кадра 320 × 240, определяет значение среднего битрейта равным 600 Кбит/с, включает кодирование в два прохода с первым проходом в турборежиме. Программа читает файл pr6.dv и выдает файл pr6.mp4. Отлично!

Кодирование WebM-видео с помощью программы ffmpeg

Я пишу эти строки 20 мая 2010 года. Формат WebM был выпущен в свет буквально вчера (я не шучу), и поэтому ассортимент программ-кодировщиков невелик, а подробных руководств совсем мало. Со временем, когда нужные инструменты будут написаны (или обновлены), запускать кодирование в WebM можно будет одним щелчком кнопкой мыши, но пока нам понадобится следующий арсенал.

- libvp8 и особая версия ffmpeg с дополнениями (обеспечивающими совместимость с libvp8), которые пока не вошли в состав официального репозитория ffmpeg:
  - установка в Linux (http://lardbucket.org/blog/archives/2010/05/19/vp8-webm-and-ffmpeg/) — эти инструкции я проверил лично под ОС Ubuntu Lucid 10.04;
  - установка в Windows (http://www.ioncannon.net/meta/1128/compiling-webm-ffmpeg-windows/) — эти инструкции я не проверял.
- Самая свежая версия mkclean (http://www.matroska.org/downloads/mkclean.html).

Готовы? Запустите ffmpeg из командной строки без параметров и удостоверьтесь, что программа была скомпилирована с поддержкой VP8:

```
you@localhost$ ffmpeg
FFmpeg version SVN-r23197, Copyright (c) 2000-2010 the FFmpeg developers
  built on May 19 2010 22:32:20 with gcc 4.4.3
  configuration: --enable-gpl --enable-version3 --enable-nonfree
  --enable-postproc --enable-pthreads --enable-libfaac --enable-libfaad
  --enable-libmp3lame --enable-libopencore-amrnb --enable-libopencore-amrwb
  --enable-libtheora --enable-libx264 --enable-libxvid --enable-x11grab
  --enable-libvpx-vp8
```

Если волшебные слова `--enable-libvpx-vp8` не появились на экране, то у вас не та версия ffmpeg, которая нам нужна. Если вы голову даете на отсечение, что все скомпилировали правильно, то проверьте, не установлено ли у вас две версии программы. Разные версии ffmpeg, установленные параллельно, не конфликтуют, но в этом случае вам надо вводить полный путь к той из них, которая поддерживает VP8.

Рассмотрим, как делать кодирование в два прохода (см. раздел «Кодирование H.264-видео с помощью HandBrake» этой главы). При первом проходе сканируется входной файл (`-i pr6.dv`) и кое-какую статистику по нему программа выводит в файл журнала, которому автоматически будет присвоено имя `pr6.dv-0.log`. Параметр `-vcodec` отвечает за выбор видеокодека:

```
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16
               -token_partitions 4 -altref 1 -lag 16 -keyint_min 0
               -g 250 -mb_static_threshold 0 -skip_threshold 0 -qmin 1 -qmax 51
               -i pr6.dv -vcodec libvpx_vp8 -an -f rawvideo -y NUL
```

Большинство этих команд никак не относится к VP8 или WebM. Библиотека libvp8 поддерживает несколько параметров, специфичных для VP8, которые можно передать ffmpeg в командной строке, но я пока не знаю, как ими пользоваться. Как только я найду в Интернете внятное объяснение сути этих параметров, на сайте оригинальной книги появится соответствующая ссылка.

При втором проходе ffmpeg прочитает статистику, записанную во время первого прохода, и выполнит собственно кодирование видео- и аудиопотоков. На выходе мы получим MKV-файл, который позднее преобразуем в WebM (в конце концов ffmpeg обязательно научится создавать WebM-файлы напрямую, но пока, по весьма тонким причинам, это невозможно).

Второй проход запускается следующими командами:

```
you you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16
                   -token_partitions 4 -altref 1 -lag 16 -keyint_min 0
                   -g 250 -mb_static_threshold 0 -skip_threshold 0
                   -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx_vp8
                   -b 614400 -s 320x240 -aspect 4:3 -acodec vorbis -y pr6.mkv
```

Здесь пять важных параметров:

- `-vcodec libvpx_vp8` — указывает, что мы пользуемся видеокодеком VP8, как и всегда при кодировании WebM-видео;
- `-b 614400` — определяет битрейт. В отличие от других инструментов, libvp8 ожидает, что пользователь введет значение в битах (а не килобитах). Таким образом, для получения видео с битрейтом 600 Кбит/с надо указать $600 \times 1024 = 614\,400$;
- `-s 320x240` — задает размер кадра в конечном видеофайле (ширину и высоту);
- `-aspect 4:3` — определяет соотношение сторон кадра. В видео обычной четкости это соотношение, как правило, равно 4:3, а в высокочетком составляет 16:9 или 16:10. Мой опыт показывает, что это соотношение надо в явном виде указать в командной строке, чтобы программа ffmpeg не ошиблась при автоопределении;

- -acodec vorbis — указывает, что мы пользуемся аудиокодеком Vorbis, как и всегда при кодировании WebM-видео.

Теперь у нас есть MKV-файл с видео VP8 и аудио Vorbis. Это уже очень близко к нужному: формат контейнеров WebM в техническом отношении очень схож с MKV, более того, он является его строгим подмножеством. Чтобы создать окончательный WebM-видеофайл, надо заменить всего лишь несколько бит с помощью названной выше утилиты mkclean:

```
you@localhost$ mkclean --doctype 4 pr6.mkv pr6.webm
```

Вот и все!

...И наконец разметка

Если это книга об HTML, то где же образцы разметки? Вот мы до них и добрались.

В HTML5 есть два способа размещения видео на веб-страницах, оба задействуют тег <video>. Если у вас только один файл, то сослаться на него можно в атрибуте src. Такая запись обнаруживает замечательное сходство с тегом для картинок <img src="...">.

```
<video src="pr6.webm"></video>
```

В принципе, больше ничего не надо указывать. Но, как и в случае с тегом <img>, рекомендуется всегда прописывать для <video> атрибуты width и height. Их значения могут быть такими же, как значения максимальной ширины и высоты кадра, выбранные вами в процессе кодирования:

```
<video src="pr6.webm" width="320" height="240"></video>
```

Не беспокойтесь, если по одной из координатных осей ваше видео чуть меньше. Браузер просто поместит его по центру прямоугольника, заданного тегом <video>, и не будет растягивать или сжимать, нарушая пропорции.

По умолчанию тег <video> не отображает виджетов, управляющих воспроизведением. Нужные кнопки можно создать с помощью старых добрых HTML, CSS и JavaScript. Для тега <video> определены такие методы, как play() и pause(), а также доступное для чтения и записи свойство currentTime; кроме этого, доступны для чтения и записи свойства volume и muted. Таким образом, в вашем распоряжении есть все необходимое, чтобы построить собственный пользовательский интерфейс.

Если не хотите строить свой интерфейс, можно воспользоваться встроенным набором элементов управления. Для этого вставьте в тег <video> атрибут controls:

```
<video src="pr6.webm" width="320" height="240" controls></video>
```

Прежде чем идти далее, скажу еще о двух необязательных атрибутах: preload и autoplay. Теперь позвольте разъяснить, чем они полезны.

Атрибут preload говорит браузеру, что скачивание видео надо начать сразу по окончании загрузки страницы. Это целесообразно в том случае, если просмотр

видео — единственная цель посещения данной страницы. Если же видео — лишь вспомогательный материал, интересный для небольшого количества посетителей, то атрибуту `preload` лучше присвоить значение `none`. Тем самым будет снижен сетевой трафик.

Вот пример веб-видео, скачивание которого (но не воспроизведение) начнется сразу после загрузки страницы:

```
<video src="pr6.webm" width="320" height="240" preload></video>
```

А вот такой клип не начнет автоматически скачиваться после загрузки страницы:

```
<video src="pr6.webm" width="320" height="240" preload="none"></video>
```

Атрибут `autoplay`, в свою очередь, говорит браузеру, что надо не только начать скачивать видео сразу по окончании загрузки страницы, но и начать его воспроизводить как можно раньше. Это, конечно, нравится не всем. Но в HTML5 важно иметь такой атрибут. Есть на свете люди, заинтересованные в том, чтобы их видеофайлы воспроизводились автоматически, невзирая на реакцию посетителей. Если бы в HTML5 не было стандартного механизма автозапуска видео, то такие люди прибегали бы к уловкам JavaScript, например вызывали метод `play()` видеоэлемента при событии `window.onload`. Этой форме навязывания очень сложно противостоять. Напротив, в нынешней ситуации можно установить (или, если угодно, написать) браузерное расширение, которое позволит всегда игнорировать атрибут `autoplay`.

Приведу пример веб-видео, скачивание и воспроизведение которого начнется, насколько это возможно, сразу после загрузки страницы:

```
<video src="pr6.webm" width="320" height="240" autoplay></video>
```

Существует сценарий Greasemonkey (http://www.greasespot.net), которым вы можете дополнить свою копию Firefox. Этот сценарий предотвращает автоматическое воспроизведение HTML5-видео. В нем используется определенный в HTML5 DOM-атрибут `autoplay`. Это JavaScript-эквивалент того атрибута `autoplay`, который доступен в HTML-коде:

```
// ==UserScript==
// @name         Disable video autoplay
// @namespace    http://diveintomark.org/projects/greasemonkey/
// @description  Ensures that HTML5 video elements do not autoplay
// @include      *
// ==/UserScript==
var arVideos = document.getElementsByTagName('video');
for (var i = arVideos.length - 1; i >= 0; i--) {
  var elmVideo = arVideos[i];
  elmVideo.autoplay = false;
}
```

Остановимся на минутку. Если вы внимательно и подряд читали эту главу, то знаете, что в нашем распоряжении сейчас не один видеофайл, а целых три. Первый — OGV-файл, созданный с помощью Firefogg (см. раздел «Кодирование Ogg-видео с помощью Firefogg» этой главы) или ffmpeg2theora (см. раздел «Пакетное

кодирование Ogg-видео с помощью ffmpeg2theora» данной главы). Второй — MP4-файл, созданный с помощью программы HandBrake (см. раздел «Кодирование H.264-видео с помощью HandBrake» этой главы). Третий — WEBM-файл, созданный с помощью ffmpeg (см. раздел «Кодирование WebM-видео с помощью ffmpeg» этой главы).

HTML5 позволяет сослаться на все три файла с помощью тега <source>. Каждый тег <video> может содержать столько тегов <source>, сколько нужно. Браузер, двигаясь сверху вниз по списку файлов-источников, воспроизведет первый же файл, который сумеет.

Возникает вопрос: как браузер определит, что может или не может воспроизвести тот или иной файл? В худшем случае он будет загружать каждый файл и пытаться воспроизвести его. Это, естественно, огромный расход трафика. Чтобы не напрягать сеть лишний раз, следует говорить браузеру всю правду о видеофайлах. Это позволяет сделать атрибут type тега <source>.

Вот как будет выглядеть дополненный код:

```
<video width="320" height="240" controls>
  <source src=”pr6.mp4”  type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src=”pr6.webm” type='video/webm; codecs="vp8, vorbis"'>
  <source src=”pr6.ogv”  type='video/ogg; codecs="theora, vorbis"'>
</video>
```

Рассмотрим его по порядку. В теге <video> указаны ширина и высота области воспроизведения, но не содержится прямой ссылки на видеофайл. Внутри <video> три тега <source>, каждый из которых посредством атрибута src ссылается на один файл и посредством атрибута type сообщает браузеру о формате этого файла. Атрибут type кажется сложным, и, в общем, не напрасно. Он содержит данные трех типов: формат контейнера (см. раздел «Видеоконтейнеры» этой главы), видеокодек (см. раздел «Видеокодеки» данной главы) и аудиокодек (см. раздел «Аудиокодеки» этой главы).

Начнем с последнего из трех файлов. Видеофайл формата OGV — это контейнер Ogg, который представлен записью video/ogg (строго говоря, это MIME-тип видеофайлов Ogg). Использованы видеокодек Theora и аудиокодек Vorbis. Здесь все достаточно просто, немного смущает лишь способ записи значения атрибута. Пара названий кодеков берется в кавычки, следовательно, значение type целиком надо взять в кавычки другого вида:

```
<source src="pr6.ogv" type='video/ogg; codecs=”theora, vorbis”'>
```

Тег <source>, отсылающий к файлу WebM, выглядит схожим образом, но оснащен другим MIME-типом (video/webm, а не video/ogg) и отсылает к другому видеокодеку (VP8, а не Theora):

```
<source src="pr6.webm" type='video/webm; codecs=”vp8, vorbis”'>
```

Тег <source>, отсылающий к файлу H.264, выглядит существенно сложнее. Если помните, я говорил, что H.264-видео (см. раздел «H.264» данной главы) и AAC-аудио (см. раздел «Advanced Audio Coding» этой главы) существуют в разных «профилях». При кодировании мы пользовались базовым профилем H.264 и профилем

низкой сложности AAC, а затем заключили видео- и аудиопоток в контейнер MPEG-4. Всю эту информацию и несет атрибут type:

```
<source src="pr6.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
```

Выгода от столь непростой записи в том, что браузер, судя лишь по атрибуту type, сумеет решить, может ли он воспроизвести тот или иной видеофайл. Если браузер сочтет, что задача ему не по силам, он *просто не станет скачивать файл* — никакую его часть. Это экономия и трафика, и времени: посетители, пришедшие на вашу страницу посмотреть видео, увидят его быстрее.

Если при кодировании видео вы последуете рекомендациям этой главы, то значения атрибута type можно будет просто скопировать из приведенного выше примера. В ином случае вам придется самостоятельно задавать параметры type.

СЕКРЕТЫ РАЗМЕТКИ

На момент написания этих строк (20 мая 2010 года) была известна ошибка в iPad, из-за которой игнорировались все видеофайлы-источники, кроме первого по порядку. Это, к сожалению, означает, что MP4-файл следует указывать первым, а уже вслед за ним — файлы открытых форматов. Эх…

MIME-типы. Целостная картина веб-видео складывается из стольких фрагментов, что я даже боюсь их все перечислять. Важно, однако, упомянуть еще один момент, связанный с неправильной конфигурацией веб-сервера. Однажды вас может весьма озадачить такое явление: на локальном компьютере видеофайлы воспроизводятся, но стоит их опубликовать на рабочем сайте, как они перестают открываться. В таком случае вину надо возлагать, скорее всего, на MIME-типы.

В главе 1 уже говорилось о MIME-типах (см. раздел «MIME-типы»), но вы, вероятно, не уделили этому достаточно внимания. Ну что ж, придется написать большими буквами:

ОЧЕНЬ ВАЖНЫЙ СЕКРЕТ

ВИДЕОФАЙЛЫ ДОЛЖНЫ ОБСЛУЖИВАТЬСЯ СООТВЕТСТВУЮЩИМИ MIME-ТИПАМИ!

Что значит «соответствующий» MIME-тип, вам уже известно: в качестве такового должна выступать часть значения атрибута type в теге <source>. Но одного лишь значения атрибута type в вашем HTML-коде недостаточно. Надо убедиться, что ваш веб-сервер включает нужный MIME-тип в HTTP-заголовок Content-Type.

Если вы пользуетесь веб-сервером Apache или одним из родственных ему, можете прописать директивы AddType в конфигурационном файле httpd.conf, действие которого распространяется на весь сайт, или в файле с расширением .htaccess в папке с вашим видео.



ПРИМЕЧАНИЕ

При пользовании другим сервером обратитесь к справочной документации. Следует узнать из нее, как назначить HTTP-заголовок Content-Type для файлов определенного типа.

Вот какие директивы AddType надо добавить:

```
AddType video/ogg .ogv
AddType video/mp4 .mp4
AddType video/webm .webm
```

Первая строка обслуживает видео в контейнере Ogg, вторая — видео в контейнере MPEG-4, а третья — WebM-видео. Вписав их, можно навсегда забыть о проблемах с MIME-типизацией видео. В противном случае в некоторых браузерах ваши видеофайлы *не будут* открываться, даже если указывать MIME-типы в атрибутах type в коде страницы.

За еще более пугающими подробностями конфигурирования веб-сервера отсылаю вас к публикации Центра Mozilla для веб-разработчиков под названием «Конфигурирование серверов под медиаданные Ogg» (https://developer.mozilla.org/en/Configuring_servers_for_Ogg_media). Советы этой статьи в равной степени применимы к видео в контейнерах MP4 и WebM.

А что в IE?

Ко времени написания книги компания Microsoft выпустила Internet Explorer 9 в «предварительной версии для разработчиков» (developer preview). Эта версия еще не поддерживает HTML5-тег <video>, но Microsoft публично пообещала (http://blogs.msdn.com/ie/archive/2010/03/16/html5-hardware-accelerated-first-ie9-platform-preview-available-for-developers.aspx), что окончательный вариант IE 9 будет поддерживать видео H.264 и аудио AAC в контейнере MPEG-4, подобно Safari на Macintosh и Mobile Safari платформы iOS.

Как же обстоит дело в более старых версиях Internet Explorer, то есть стабильных версиях до восьмой включительно? Большинство пользователей Internet Explorer пользуются также приложением Adobe Flash, современные версии которого (начиная с 9.0.60.184) поддерживают видео H.264 и аудио AAC в контейнере MPEG-4. Таким образом, H.264-видео, закодированное вами для Safari (см. раздел «Кодирование H.264-видео с помощью HandBrake» этой главы), может быть воспроизведено с помощью видеопроигрывателя на основе Flash, если система обнаружит, что браузер посетителя не поддерживает HTML5-функциональность.

Flowplayer (http://flowplayer.org) — видеопроигрыватель на основе Flash с открытым исходным кодом, распространяемый под лицензией GPL (для него доступны и коммерческие лицензии, о которых рассказано по адресу http://flowplayer.org/download/). О теге <video> проигрывателю Flowplayer ничего не известно, и магическим образом преобразовывать тег <video> во Flash-объект он не умеет. Но это легко поправимо средствами HTML5, ведь внутрь <video> можно вложить тег <object>. Браузеры без поддержки HTML5-видео будут игнорировать тег <video> и отображать вместо него вложенный <object>, который, вызывая Flash, будет воспроизводить видео с помощью Flowplayer. Браузеры же с поддержкой HTML5-видео отыщут видеофайл-источник и воспроизведут его, тогда как вложенный <object> *будет полностью проигнорирован.*

Последнее замечание — ключ ко всему остальному. В HTML5 все дочерние узлы тега <video>, кроме <source>, полностью игнорируются. Это позволяет применять в более новых браузерах HTML5-видео, а в более старых — элегантный запасной вариант, Flash. Никаких трюков на JavaScript не требуется. Подробнее об этом способе можно прочесть на сайте http://camendesign.com/code/video_for_everybody в статье «Видео для каждого».

Живой пример

Рассмотрим образец, использующий данную технологию. Я дополнил код из статьи «Видео для каждого» так, чтобы стало возможным вставить на страницу клип в формате WebM. Вот команды терминала для кодирования одного и того же исходного файла в три разных формата:

```
## Theora/Vorbis/Ogg
you@localhost$ ffmpeg2theora --videobitrate 200 --max_size 320x240
               --output pr6.ogv pr6.dv
## H.264/AAC/MP4
you@localhost$ HandBrakeCLI --preset “iPhone & iPod Touch” --vb 200 --width 320
               --two-pass --turbo --optimize --input pr6.dv --output pr6.mp4
## VP8/Vorbis/WebM
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16
               -token_partitions 4 -altref 1 -lag 16 -keyint_min 0 -g 250
               -mb_static_threshold 0 -skip_threshold 0 -qmin 1 -qmax 51
               -i pr6.dv -vcodec libvpx_vp8 -an -f rawvideo -y NULffmpeg
               --videobitrate 200
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16
              -token_partitions 4 -altref 1 -lag 16 -keyint_min 0 -g 250
              -mb_static_threshold 0 -skip_threshold 0 -qmin 1 -qmax 51
              -i pr6.dv -vcodec libvpx_vp8 -b 204800 -s 320x240 -aspect 4:3
              -acodec vorbis -ac 2 -y pr6.mkv
you@localhost$ mkclean --doctype 4 pr6.mkv pr6.webm
```

В окончательном коде страницы используется тег <video> для HTML5-видео, а вложенный в него <object> — для запасного варианта с привлечением Flash:

```
<video id="movie" width="320" height="240" preload controls>
  <source src="pr6.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"' />
  <source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"' />
  <source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"' />
  <object width="320" height="240" type="application/x-shockwave-flash"
     data="flowplayer-3.2.1.swf">
    <param name="movie" value="flowplayer-3.2.1.swf" />
    <param name="allowfullscreen" value="true" />
    <param name="flashvars" value='config={
      "clip": {"url": "http://wearehugh.com/dih5/good/bbb_480p.mp4",
      "autoPlay":false, "autoBuffering":true}}' />
    <p>Скачать видео в формате <a href="pr6.mp4">MP4</a>,
      <a href="pr6.webm">WebM</a>, or
```

```
    <a href="pr6.ogv">Ogg</a>.</p>
  </object>
</video>
```

Сочетание HTML5 и Flash позволяет просматривать это видео практически в любом браузере на любом устройстве.

Для дальнейшего изучения

- Тег <video> в спецификации HTML5 (http://bit.ly/a3kpiq).
- «Видео для каждого» (http://camendesign.com/code/video_for_everybody).
- «Элементарное введение в кодировки видеоданных» (http://diveintomark.org/tag/give).
- «Что следует знать о новом кодеке Theora 1.1» (http://hacks.mozilla.org/2009/09/theora-1-1-released/) — статья Кристофера Близзарда (Christopher Blizzard).
- «Конфигурирование серверов под медиаданные Ogg» (https://developer.mozilla.org/en/Configuring_servers_for_Ogg_media).
- «Кодирование с помощью кодека x264» (http://www.mplayerhq.hu/DOCS/HTML/en/menc-feat-x264.html).
- «Свойства разных типов видео» (http://wiki.whatwg.org/wiki/Video_type_parameters).
- Zencoder Video JS (http://videojs.com) — нестандартные элементы управления HTML5-видео.
- «Все, что вам нужно знать об аудио и видео в HTML5» (http://dev.opera.com/articles/view/everything-you-need-to-know-about-html5-video-and-audio/) — статья Саймона Питерса (Simon Pieters).

6 Вот мы вас и нашли!

Приступим

Геолокация — это такой вид деятельности, который позволяет определять координаты своего местонахождения в мире и при желании делиться ими с друзьями. Координаты можно определить несколькими способами: по IP-адресу, через подключение к беспроводному Интернету, через ретранслятор сотовой сети, на связи с которым находится ваш телефон, или с помощью особого GPS-аппарата, принимающего спутниковые данные о широте и долготе.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Я боюсь геолокации. Можно ли ее выключить?

Ответ: Вполне естественно, что при передаче на сервер данных о физическом местонахождении встает проблема конфиденциальности. В стандарте API геолокации прямо говорится (http://www.w3.org/TR/geolocation-API/#security): «Пользовательские агенты не должны без явного разрешения пользователя отсылать сайтам геолокационные сведения». Иными словами, не хотите — не рассказывайте, где вы сейчас.

API геолокации

Делиться своим местонахождением с надежными сайтами позволяет API геолокации. К значениям широты и долготы получает доступ JavaScript, который сможет переслать эту информацию обратно на веб-сервер, а тот, в свою очередь, как-либо ее обработает: покажет ваше положение на карте, подыщет вам, чем заняться на досуге, и т. д.

Как можно видеть из табл. 6.1, API геолокации уже поддерживается в некоторых популярных браузерах на десктопных и мобильных платформах. В помощь более старым браузерам созданы библиотеки, о которых речь пойдет далее в этой главе.

Таблица 6.1. Поддержка API геолокации

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
–	3.5+	5.0+	5.0+	–	3.0+	2.0+

Кроме стандартного API геолокации, разные мобильные платформы предусматривают собственные API. О них мы тоже еще поговорим.

Покажите мне код

Основополагающим для API геолокации считается новое свойство navigator.geolocation глобального объекта navigator.

Простейшая функция для работы с API геолокации такова:

```
function get_location() {
  navigator.geolocation.getCurrentPosition(show_map);
}
```

Эта функция не проверяет поддержку API, не обрабатывает ошибки и не принимает никаких параметров. Но в хорошем веб-приложении дело должно обстоять совсем наоборот хотя бы с первыми двумя пунктами из названных трех.

Для тестирования поддержки API геолокации (см. раздел «Геолокация» главы 2) воспользуйтесь Modernizr:

```
function get_location() {
  if (Modernizr.geolocation) {
    navigator.geolocation.getCurrentPosition(show_map);
  } else {
    // нет встроенной поддержки, можно попробовать Gears
  }
}
```

Что делать, если геолокация не поддерживается, — необходимо решать веб-мастеру. Как воспользоваться Gears в качестве резервного варианта, я скоро расскажу, но перед этим надо объяснить, как обрабатывается вызов функции getCurrentPosition(). В начале этой главы я отметил, что поддержка геолокации не является принудительной. Браузер никогда не заставит вас передать на веб-сервер данные о местонахождении помимо вашей воли. На вызов функции getCurrentPosition() геолокационного API различные браузеры реагируют по-разному, но общая логика едина. Так, в Mozilla Firefox в верхней части окна всплывет «информационная панель». Как она выглядит, показано на рис. 6.1.

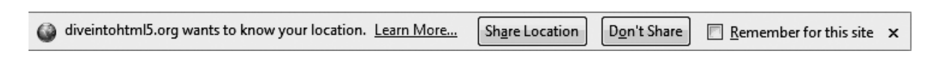


Рис. 6.1. Панель геолокации

Конечному пользователю эта панель сообщает:

- что сайт запрашивает его местонахождение;
- *какой* именно сайт запрашивает его местонахождение.

Пользователь может:

- перейти на справочную страницу Mozilla http://www.mozilla.com/en-US/firefox/geolocation/, чтобы понять суть происходящего;

- разрешить передачу геолокационных данных;
- *запретить* передачу геолокационных данных;
- велеть браузеру запомнить выбор (разрешение/запрет), чтобы при последующих посещениях того же сайта панель больше не всплывала.

Кроме того, описываемая панель:

- немодальна, то есть позволяет переключаться между вкладками и окнами браузера;
- привязана к определенной вкладке, то есть переход к другой вкладке или окну браузера заставит ее исчезнуть, а если вернуться на прежнюю вкладку, она вновь появится;
- безусловна, то есть сайт не может ее обойти;
- блокирует запрос, то есть, пока браузер ожидает вашего ответа, сайт не имеет возможности определить ваше местонахождение.

Выше был показан код на JavaScript, в результате обработки которого браузер выводит панель. Это функция, принимающая в качестве аргумента пользовательскую функцию обратного вызова `show_map()`. Вызов `getCurrentPosition()` обрабатывается сразу, но доступ к данным о местонахождении пользователя сценарий получит не сразу. Когда они будут доступны, к ним прежде всего получит доступ функция обратного вызова. Она может выглядеть примерно так:

```
function show_map(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  // покажем вас на карте или сделаем что-нибудь еще!
}
```

Эта функция вызывается с единственным аргументом, который представляет собой объект с двумя свойствами: `coords` и `timestamp`. Свойство `timestamp` хранит дату и время расчета координат.



ПРИМЕЧАНИЕ

Поскольку все происходит асинхронно, нельзя точно предугадать момент расчета. Пользователю может потребоваться какое-то время, чтобы прочесть предупреждение и согласиться на отсылку геолокационных данных; мобильное устройство с GPS-модулем должно будет еще связаться со спутником и т. д.

У объекта `coords` есть несколько собственных свойств: `latitude` (широта), `longitude` (долгота) и др. Их значения отвечают названиям. Все они в совокупности представляют физическое местоположение пользователя в мире.

Свойства объекта `position` показаны в табл. 6.2.

Таблица 6.2. Свойства объекта position

Свойство	Тип	Комментарий
coords.latitude	double	Десятичные градусы
coords.longitude	double	Десятичные градусы

Свойство	Тип	Комментарий
coords.altitude	double или null	Метры над поверхностью условного эллипсоида Земли
coords.accuracy	double	Метры
coords.altitudeAccuracy	double или null	Метры
coords.heading	double или null	Градусы по часовой стрелке от направления на север
coords.speed	double или null	Метры в секунду
timestamp	DOMTimeStamp	Как объект Date()

Гарантированно определены только три координатных свойства: `coords.latitude`, `coords.longitude` и `coords.accuracy`. Представление остальных (`null`, если не определены) зависит от возможностей клиентского компьютера и сервера позиционирования, с которым тот связывается. Свойства `heading` и `speed` могут быть вычислены лишь тогда, когда известно предыдущее положение пользователя.

Обработка ошибок

Геолокация — это непростая система, в которой того и гляди что-нибудь не заладится. Уже говорилось о проблеме запроса согласия пользователя. Если страница запрашивает координаты пользователя, а пользователь не хочет их сообщать, веб-мастер будет посрамлен, потому что пользователь всегда прав. Каким же образом реализовать в коде обработку ошибок? Для нее предназначен второй аргумент функции `getCurrentPosition()` — это тоже функция обратного вызова:

```
navigator.geolocation.getCurrentPosition(show_map, handle_error)
```

Если что-то не в порядке, функция обработки ошибок будет применена к объекту `PositionError`. Его свойства перечислены в табл. 6.3.

Таблица 6.3. Свойства объекта PositionError

Свойство	Тип	Комментарий
code	short	Код ошибки (порядковый номер)
message	DOMString	Не для конечных пользователей

Свойство `code` принимает одно из четырех значений:

- `PERMISSION_DENIED` (1) — если пользователь нажмет кнопку Don't Share (Не сообщать) на панели геолокации или иным способом запретит доступ к данным о своем местонахождении;
- `POSITION_UNAVAILABLE` (2) — если не работает сеть или невозможно связаться со спутником позиционирования;
- `TIMEOUT` (3) — когда сеть работает, но на расчет координат уходит слишком много времени (что значит «слишком много», вы узнаете в следующем разделе);
- `UNKNOWN_ERROR` (0) — если проблема в чем-то другом.

Пример:

```
function handle_error(err) {
  if (err.code == 1) {
    // воля пользователя — закон для меня!
  }
}
```

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Будет ли API геолокации работать на Международной космической станции, на Луне, на других планетах Солнечной системы?

Ответ: Спецификация (http://www.w3.org/TR/geolocation-API/#coordinates_interface) гласит: «Эталонная система географических координат, используемая в атрибутах этого интерфейса, — Мировая геодезическая система [WGS84] с двухмерными координатами. Другие системы не поддерживаются». Поскольку МКС облетает Землю по орбите, местоположение космонавтов на станции (http://twitter.com/Astro_TJ) описывается широтой, долготой и высотой. Но, будучи ориентированной на координатную сеть Земли, Мировая геодезическая система неприменима к Луне и другим планетам.

Требую выбора!

Такие популярные мобильные устройства, как iPhone и Android, поддерживают *два* способа определения координат. Первый способ — триангуляция, основанная на оценке расстояний до нескольких башен-ретрансляторов вашего сотового оператора. Это быстрая процедура, для которой не нужно никакого специального GPS-оборудования, но ее точность оставляет желать лучшего. В зависимости от того, сколько ретрансляторов находится поблизости, ваше местонахождение может быть установлено с точностью до квартала (если их много) или до одного-двух километров (если их мало).

Другой способ состоит в том, что GPS-модуль вашего мобильного устройства связывается с находящимися на околоземной орбите спутниками GPS-позиционирования. Как правило, GPS позволяет определить местонахождение с точностью до нескольких метров. Обратная сторона медали — чрезвычайно высокое энергопотребление GPS-чипов. На мобильном устройстве общего назначения такой модуль, как правило, по умолчанию выключен и включается лишь при необходимости. Вот почему инициализация связи с GPS-спутником требует некоторого времени.

Если вам когда-либо приходилось открывать Карты Google на iPhone или другом смартфоне, то оба способа вам уже знакомы. Сначала на карте появляется большой круг (окрестность ближайшей башни-ретранслятора), затем он уменьшается (триангуляция по другим башням) и наконец стягивается в точку (координаты, полученные с GPS-спутника).

Я рассказываю об этом потому, что не каждое веб-приложение нуждается в высокой точности координат. Так, например, если пользователь ищет, какой бы ему фильм посмотреть в одном из окрестных кинотеатров, хватит «приблизительных» координат, ведь даже в больших городах кинотеатров не так много и так или иначе

система должна будет вывести афиши всего нескольких из них. С другой стороны, для пошагового инструктирования в реальном времени нужна самая высокая точность. Иначе невозможно будет сказать пользователю: «Пройдя 20 метров, сверните направо» и т. п.

У функции `getCurrentPosition()` есть необязательный третий аргумент — объект `PositionOptions`. В нем можно устанавливать значения трех свойств (табл. 6.4). Свойства объекта `PositionOptions` тоже необязательны.

Таблица 6.4. Свойства объекта PositionOptions

Свойство	Тип	По умолчанию	Комментарий
enableHighAccuracy	boolean	false	Установка true может замедлить работу
timeout	long	(не установлено)	В миллисекундах
maximumAge	long	0	В миллисекундах

Если функциональность устройства позволяет рассчитывать точные координаты, а пользователь согласен делиться ими, то их пересылку на сервер можно включить, установив для свойства `enableHighAccuracy` значение `true`. На смартфонах iPhone и Android доступ к «низкоточному» и «высокоточному» позиционированию разделен, поэтому возможно, что вызов `getCurrentPosition()` при `enableHighAccuracy:true` выдаст ошибку, а при `enableHighAccuracy:false` пройдет успешно.

Свойство `timeout` определяет длительность промежутка (в миллисекундах), в течение которого ваше веб-приложение будет ждать присылки координат. Отсчет времени начинается с того момента, когда пользователь разрешает браузеру передать сведения о своем местонахождении. Таким образом, мы ограничиваем во времени не пользователя, а сеть.

Свойство `maximumAge` позволяет устройству сразу отсылать кэшированные данные о координатах. Так, например, пусть ваша страница вызвала `getCurrentPosition()` в первый раз, пользователь согласился и ваша первая (на случай успеха) функция обратного вызова применяется к координатам, вычисленным ровно в 10:00 утра. Допустим, через минуту после этого, в 10:01, вы снова вызываете `getCurrentPosition()`, на этот раз с присвоенным атрибуту `maximumAge` значением `75000`:

```
navigator.geolocation.getCurrentPosition(
success_callback, error_callback, {maximumAge: 75000});
```

Написав так, вы заявляете, что собственно *нынешнее* местонахождение пользователя вам знать не обязательно; достаточно узнать, где он находился 75 секунд (75 000 миллисекунд) назад. В данном случае устройство помнит, где пользователь был 60 секунд (60 000 миллисекунд) назад: координаты вычислялись при первом вызове `getCurrentPosition()`. Поскольку эти сведения не старше оговоренного срока давности, система не выполняет повторный расчет координат, а возвращает те же самые данные, что и в первый раз: широту, долготу, точность, метку времени (10:00 утра).

Итак, прежде чем запрашивать пользователя о его местонахождении, задумайтесь о том, какая вам нужна точность данных, и, если это необходимо, включите `enableHighAccuracy`. Если местонахождение пользователя надо будет определять

несколько раз, продумайте целесообразный срок давности сведений и установите соответствующий maximumAge. В том случае, если за положением пользователя надо следить непрерывно, функция getCurrentPosition() вам не подойдет. Надо применить watchPosition().

У watchPosition() та же структура, что и у getCurrentPosition(). В качестве аргументов она принимает две функции обратного вызова: одну обязательную (на случай успешного вызова) и одну необязательную (на случай ошибки), а необязательный третий аргумент — объект PositionOptions, со свойствами которого вы познакомились ранее. Разница в том, что ваша функция обратного вызова будет выполняться *каждый раз, когда местонахождение пользователя изменится*. Активным образом «опрашивать» аппарат пользователя нет необходимости: он сам определит удобный интервал «опроса» и будет запускать первую из двух функций обратного вызова каждый раз после того, как координаты пользователя изменятся. Можно применять watchPosition() для отображения движущегося маркера на карте, построения маршрута в реальном времени и т. д. по вашему усмотрению.

Функция watchPosition() возвращает число, которое вы, вероятно, посчитаете нужным где-то хранить. Чтобы прекратить слежение за координатами пользователя, достаточно вызвать метод clearWatch() и передать ему это число. В результате аппарат пользователя прекратит запускать функцию обратного вызова. Если вам когда-либо приходилось применять функции setInterval() и clearInterval() в JavaScript, знайте, что здесь механизм точно такой же.

А что в IE?

API геолокации, стандартизованный W3C (см. раздел «API геолокации» этой главы), не поддерживается в Internet Explorer. Но не спешите отчаиваться! Gears (http://tools.google.com/gears/) — разработанное Google браузерное расширение с открытым исходным кодом, функционирующее на платформах Windows, Mac OS X, Linux, Windows Mobile и Android. Оно эмулирует в старых браузерах некоторые из новых функций, в частности API геолокации. Этот интерфейс отличается от стандартизованного W3C, но служит тем же целям.

Раз уж мы заговорили о наследуемой функциональности платформ, отмечу, что мобильные аппараты, построенные на платформах BlackBerry, Nokia, Palm и OMTP BONDI, имеют собственные API геолокации. Разумеется, все эти прикладные интерфейсы отличаются от интерфейса Gears, который, в свою очередь, отличается от W3C API геолокации. Как же быть?..

На помощь спешит geo.js

Сценарий geo.js (http://code.google.com/p/geo-location-javascript/) — это JavaScript-библиотека, распространяемая под лицензией MIT с открытым исходным кодом. Она позволяет сглаживать различия между W3C API геолокации, API приложения Gears и разнообразными API мобильных платформ. Чтобы воспользоваться geo.js,

надо поместить в нижнюю часть страницы два сценария (строго говоря, их можно поместить в любую часть страницы, но если внедрить их в контейнер <head>, страница будет загружаться медленнее, так что не делайте этого).

Первый из этих сценариев — gears_init.js (**http://code.google.com/apis/gears/gears_init.js**), инициализирующий Gears, если это расширение установлено. Второй — geo.js (**http://geo-location-javascript.googlecode.com/svn/trunk/js/geo.js**). Код страницы должен выглядеть приблизительно так:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Название моей страницы</title>
  </head>
  <body>
    ...
    <script src="gears_init.js"></script>
    <script src="geo.js"></script>
  </body>
</html>
```

Теперь вы готовы к использованию любого из доступных API геолокации:

```
if (geo_position_js.init()) {
  geo_position_js.getCurrentPosition(geo_success, geo_error);
}
```

Разберем этот код по порядку. Сначала надо явным образом вызвать функцию init(). Если поддерживается и доступен хотя бы один API геолокации, функция возвратит true:

```
if (geo_position_js.init()) {
```

Вызвав функцию init(), мы не устанавливаем местонахождение пользователя, а только удостоверяемся, что это возможно сделать. Чтобы перейти к делу, вызовем функцию getCurrentPosition():

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

При вызове getCurrentPosition() браузер спросит пользователя, можно ли рассчитать и передать на сайт его координаты. В частности, при геолокации с помощью Gears появится окно с вопросом, разрешает ли пользователь сайту прибегнуть к функциональности Gears. При встроенной поддержке API геолокации в браузере окно будет выглядеть иначе (например, в Firefox 3.5, как вы уже видели, появляется информационная панель в верхней части страницы).

Функция getCurrentPosition() принимает в качестве аргументов две функции обратного вызова. Если с помощью getCurrentPosition() местонахождение пользователя было успешно найдено (то есть пользователь согласился на передачу данных и API геолокации благополучно исполнил запрос), то выполняется первая из переданных функций. В данном примере она называется geo_success:

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

Эта функция обратного вызова, обслуживающая ситуацию «успеха», принимает единственный аргумент — объект с информацией о местонахождении:

```
function geo_success(p) {
  alert("Вы нашлись в точке с долготой " + p.coords.latitude +
        " и широтой " + p.coords.longitude);
}
```

Если функция getCurrentPosition() не сумела установить координаты пользователя (из-за того что он отказался посылать их, или из-за какого-либо сбоя в API геолокации), то будет вызвана функция, переданная ей как второй аргумент. В данном примере функция обратного вызова, обслуживающая ситуацию «неудачи», называется geo_error:

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

Она не принимает никаких аргументов:

```
function geo_error() {
  alert("Вы не нашлись!");
}
```

Поддержки функции watchPosition() в geo.js пока нет. Чтобы постоянно следить за местонахождением пользователя, надо активным образом «опрашивать» его устройство, вызывая getCurrentPosition().

Живой пример

Изучим следующий пример использования geo.js для нахождения координат пользователя и вывода на экран карты местности, в которой он находится.

При загрузке страницы надо вызвать geo_position_js.init(), чтобы выяснить, доступен ли хотя бы один интерфейс геолокации из тех, которые поддерживает geo.js. Если API доступен, то можно вывести ссылку, щелкнув на которой, пользователь сможет увидеть свое местоположение на карте. Переход по ссылке будет вызывать функцию lookup_location() следующего вида:

```
function lookup_location() {
  geo_position_js.getCurrentPosition(show_map, show_map_error);
}
```

Если пользователь соглашается на пересылку данных, а геолокационному сервису удается установить его координаты, то geo.js запустит первую функцию обратного вызова — show_map() — с единственным аргументом loc. Свойство coords объекта loc содержит широту, долготу и сведения о точности измерений (в этом примере точность измерений не учитывается). Затем функция show_map() с помощью API Карты Google выводит на страницу карту с маркером:

```
function show_map(loc) {
  $("#geo-wrapper").css({'width':'320px','height':'350px'});
  var map = new GMap2(document.getElementById("geo-wrapper"));
  var center = new GLatLng(loc.coords.latitude, loc.coords.longitude);
```

```
  map.setCenter(center, 14);
  map.addControl(new GSmallMapControl());
  map.addControl(new GMapTypeControl());
  map.addOverlay(new GMarker(center, {draggable: false, title: "Вы нашлись
                                    приблизительно здесь"}));
}
```

Если geo.js не может определить местоположение пользователя, будет вызвана вторая функция обратного вызова — show_map_error():

```
function show_map_error() {
  $("#live-geolocation").html('Вы не нашлись!');
}
```

Для дальнейшего изучения

- Стандарт API геолокации на сайте W3C (http://www.w3.org/TR/geolocation-API/).
- Gears (http://tools.google.com/gears/).
- API геолокации платформы BlackBerry (http://www.tonybunce.com/2008/05/08/Blackberry-Browser-Amp-GPS.aspx).
- API геолокации платформы Nokia (http://wiki.forum.nokia.com/index.php/Bondi_Widget_porting_example_-_geolocation_API).
- API геолокации платформы Palm (http://developer.palm.com/index.php?option=com_content&view=article&id=1673#GPS-getCurrentPosition).
- API геолокации платформы OMTP BONDI (http://bondi.omtp.org/1.0/apis/geolocation.html).
- geo.js — сценарий-обертка для разных геолокационных API (http://code.google.com/p/geo-location-javascript/).

7 Локальное хранилище: прошлое, настоящее, будущее

Приступим

Постоянное локальное хранилище данных — одна из областей, в которой обычные клиентские приложения традиционно имели преимущество перед веб-приложениями. Для первых операционная система, как правило, предоставляет такой уровень абстракции, где можно хранить собственные данные приложений (например, пользовательские настройки и состояние) и откуда эти данные можно запрашивать. В качестве хранилища могут выступать реестр, INI-файлы, XML-файлы или что-либо еще в зависимости от правил системы. Если клиентская программа нуждается в хранении данных иного типа, нежели пары «ключ — значение», то можно встроить в нее (программу) свою собственную базу данных, придумать собственный формат файлов или реализовать какое-либо из других возможных решений.

Всей этой роскошью веб-приложения исторически не располагали. Cookies были изобретены довольно рано, и в роли постоянного локального хранилища для небольших порций данных они, конечно, могут применяться. Но у них есть несколько неприятных особенностей.

- Cookies входят в состав каждого HTTP-запроса и поэтому замедляют работу вашего веб-приложения, которое вынуждено снова и снова бесцельно передавать одни и те же данные.
- Поскольку cookies входят в состав каждого HTTP-запроса, они посылают данные через Интернет в незашифрованном виде (если только все веб-приложение не шифруется с помощью SSL).
- По объему cookies ограничены приблизительно 4 Кбайт. Этого достаточно, чтобы замедлить работу приложения (см. выше), но недостаточно, чтобы принести взамен какую-либо ощутимую выгоду.

Нам нужно совсем другое, а именно:

- большое по объему хранилище...
- на клиентской стороне...
- данные которого не меняются при обновлении страницы...
- и не передаются на сервер.

Известно о нескольких попытках воплотить этот идеал, но до недавнего времени все они оказывались в конечном счете неудачными — и притом по-разному неудачными.

Краткая история прототипов локального хранилища до HTML5

В начале было не Слово, а Internet Explorer. По крайней мере разработчики Microsoft хотят, чтобы все думали именно так. Надо отдать должное специалистам Microsoft: во время первой из «великих войн браузеров» они придумали множество новинок, которые и включили в свой браузер-победитель Internet Explorer. Одна из новинок называлась DHTML Behaviors; в числе форм «поведения» браузера была одна под названием `UserData`.

Форма `UserData` разрешает веб-страницам хранить до 64 Кбайт данных на каждый домен в иерархической структуре на основе XML. (Доверенные сайты, в частности интранет-сайты, могут хранить в десять раз больше данных. Да-да, еще недавно считалось, что 640 Кбайт хватит всем и на любые нужды!) Никакого диалога об увеличении объема хранимых данных IE вести не умел, поэтому способа расширить хранилище не существовало.

В 2002 году специалисты Adobe ввели во Flash 6 функцию, которая была неудачно и обманчиво названа Flash-cookies. В среде Flash эта функция известна под названием Local Shared Objects (распределенные локальные объекты, или LSO). Если говорить кратко, то LSO позволяет Flash-объектам хранить до 100 Кбайт данных на каждый домен. В 2005 году Брэд Ньюберг (Brad Neuberg) разработал прототип системы взаимодействия Flash с JavaScript, которая получила название AMASS (AJAX Massive Storage System). Некоторые конструктивные особенности Flash ограничивали работу AMASS, но уже в 2006 году, после появления ExternalInterface во Flash 8, оперировать LSO с помощью JavaScript стало на порядок легче и быстрее. Тогда Брэд, переписав AMASS, ввел ее в состав популярного набора инструментов Dojo Toolkit под названием `dojox.storage`. При использовании `dojox.storage` каждый домен может безвозмездно получить от Flash стандартные 100 Кбайт на хранение данных. Более того, пользователь может последовательно увеличивать объем хранимых данных на порядок (до 1 Мбайт, затем до 10 Мбайт и т. д.).

В 2007 году компания Google создала Gears — браузерное расширение с открытым исходным кодом, придающее браузерам дополнительную функциональность (говоря об API геолокации в Internet Explorer, я уже упоминал Gears: см. раздел «А что в IE?» главы 6). Gears предоставляет API для встроенной базы данных на основе SQLite. Один раз получив согласие пользователя, Gears сможет хранить неограниченное количество данных для каждого домена в таблицах базы данных SQL.

Тем временем Брэд Ньюберг и его коллеги продолжали работать над `dojox.storage`. Их целью было обеспечить единый интерфейс для всех существующих приложений и API. Уже к 2009 году система `dojox.storage` была способна автоматически распознавать (и оснащать единым интерфейсом доступа) Adobe Flash, Gears, Adobe AIR и один прообраз хранилища HTML5, реализованный только в старых версиях Firefox.

Читая обо всех этих решениях, вы наверняка подметили некую закономерность: каждое из них или привязано к какому-то одному браузеру или зависит от стороннего приложения. Несмотря на героические усилия команды `dojox.storage` по наведению мостов, все имеющиеся альтернативы радикально различны по своим интерфейсам, имеют разные ограничения на объем хранимых данных и по-разному взаимодействуют с пользователем. Итак, в HTML5 должна была быть решена проблема следующего вида: нужно создать стандартный API, полная поддержка которого без опоры на сторонние приложения была бы реализована в нескольких браузерах.

HTML5-хранилище: вводный курс

Рассматриваемое здесь HTML5-хранилище, описано в спецификации Web Storage, которая одно время являлась частью спецификации HTML5, но затем была выделена в особую спецификацию (по причинам политического характера, неинтересным для нас). Некоторые разработчики браузеров пользуются терминами «локальное хранилище» и «DOM-хранилище». Ситуацию с названием еще осложняют родственные развивающиеся стандарты, названные схожим образом; я скажу о них далее в этой главе.

Что же такое HTML5-хранилище? Если говорить просто, то это способ хранить создаваемые веб-страницами именованные пары «ключ — значение» локально, в клиентской программе, то есть в браузере. Эти данные, как и cookies, будут храниться даже после того, как пользователь покинет сайт, закроет вкладку браузера или вообще выйдет из браузера и пр. Но, в отличие от cookies, эти данные никогда не пересылаются на удаленный веб-сервер, если только сам пользователь не пожелает отправить их вручную. В отличие от более ранних образцов постоянного локального хранилища, о которых идет речь в предыдущем разделе, нынешняя технология встроена в браузеры и доступна без сторонних приложений.

Уместен вопрос: в каких браузерах поддерживается HTML5-хранилище? В табл. 7.1 показано, что таковы последние версии всех основных браузеров. Среди них даже Internet Explorer!

Таблица 7.1. Поддержка HTML5-хранилища

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
8.0+	3.5+	4.0+	4.0+	10.5+	2.0+	2.0+

В коде на JavaScript доступ к HTML5-хранилищу обеспечивает объект `localStorage` глобального объекта `window`. Прежде чем пользоваться технологией, убедитесь, что браузер поддерживает ее (см. раздел «Локальное хранилище» главы 2):

```
function supports_html5_storage() {
  return ('localStorage' in window) && window['localStorage'] !== null;
}
```

Чтобы не писать собственную функцию для тестирования поддержки HTML5-хранилища, прибегните к помощи Modernizr (см. раздел «Modernizr: библиотека для тестирования HTML5-функций» главы 2):

```
if (Modernizr.localstorage) {
  // хранилище доступно!
} else {
  // встроенной поддержки хранилища нет,
  // стоит попробовать dojox.storage или иное стороннее решение
}
```

Использование HTML5-хранилища

В основе HTML5-хранилища лежат именованные пары «ключ — значение». Вы сохраняете данные под именованным ключом, а затем по тому же ключу система будет возвращать вам эти данные:

```
interface Storage {
  getter any getItem(in DOMString key);
  setter creator void setItem(in DOMString key, in any data);
};
```

Сам именованный ключ — обязательно строка. Данные, сохраняемые под ключом, могут быть любого типа, поддерживаемого JavaScript: строки, логические значения, целые числа, числа с плавающей запятой. Впрочем на самом деле любые данные хранятся в виде строк. Если вы сохраняете и извлекаете не строки, то для перевода извлеченных данных в ожидаемый JavaScript-тип надо будет пользоваться функциями `parseInt()`, `parseFloat()` и им подобными.

При вызове `setItem()` с именем ключа, которому уже присвоено какое-либо значение, это значение будет заменено новым. Если вызвать `getItem()` с несуществующим ключом, то система не возбудит исключение, а возвратит `null`.

Объект `localStorage`, подобно некоторым другим объектам JavaScript, можно рассматривать как ассоциативный массив. Это значит, что вместо методов `getItem()` и `setItem()` разрешается пользоваться квадратными скобками. Рассмотрим, например, следующий фрагмент кода:

```
var foo = localStorage.getItem("bar");
// …
localStorage.setItem("bar", foo);
```

Если применить синтаксис с квадратными скобками, то этот фрагмент примет следующий вид:

```
var foo = localStorage[“bar”];
// …
localStorage[“bar”] = foo;
```

Есть также методы, позволяющие удалять значения именованных ключей и вообще очищать хранилище (то есть сразу удалять все ключи и значения):

```
interface Storage {
  deleter void removeItem(in DOMString key);
  void clear();
}
```

Если применить removeItem() к несуществующему ключу, то ничего не произойдет.

Наконец, имеется свойство, позволяющее узнавать общее количество значений в хранилище и перебирать все ключи по индексу (то есть определять имя каждого ключа):

```
interface Storage {
  readonly attribute unsigned long length;
  getter DOMString key(in unsigned long index);
};
```

Если при вызове key() индекс лежит не в диапазоне от 0 до length-1, то функция вернет null.

Следим за состоянием HTML5-хранилища

Если вашей программе нужно следить за изменениями в состоянии хранилища, то она должна реагировать на событие storage. Событие storage объекта window происходит, когда после вызова setItem(), removeItem() или clear() что-либо в хранилище меняется. Так, если вы вновь присвоили ключу уже закрепленное за ним значение или вызвали clear() в момент, когда хранилище пусто, storage не сработает, потому что в области хранения ничего не изменилось.

Событие storage поддерживается везде, где поддерживается объект localStorage, то есть и в Internet Explorer 8. Но в IE 8 не реализован стандарт W3C addEventListener (который, правда, будет наконец-то добавлен в IE 9). Значит, чтобы ваш код перехватывал событие storage, надо узнать, какой механизм работы с событиями поддерживает браузер[1]. Это можно сделать так:

```
if (window.addEventListener) {
  window.addEventListener("storage", handle_storage, false);
} else {
  window.attachEvent("onstorage", handle_storage);
};
```

Функция обратного вызова handle_storage будет применена к объекту StorageEvent, а в Internet Explorer — к хранящему события свойству window.event:

```
function handle_storage(e) {
  if (!e) { e = window.event; }
}
```

После выполнения этого фрагмента кода переменная e станет объектом StorageEvent. Его полезные свойства перечислены в табл. 7.2.

[1] Подготовленный читатель, который уже знает, как перехватывать события, может пропустить весь этот раздел. Перехват события storage работает стандартным образом. В частности, если для регистрации обработчиков событий вы пользуетесь jQuery или какой-либо другой библиотекой JavaScript, то их возможности можно распространить и на storage. — *Примеч. авт.*

Таблица 7.2. Свойства объекта StorageEvent

Свойство	Тип	Описание
key	Строка	Имя ключа, значение которого было добавлено, удалено или изменено
oldValue	Любой	Предыдущее значение ключа (если удалено, изменено) или null (если добавлено новое значение)
newValue	Любой	Новое значение ключа (если добавлено, изменено) или null (если значение удалено)
url*	Строка	Адрес страницы, которая вызвала метод, изменивший состояние хранилища

* Свойство url первоначально называлось uri. Прежде чем спецификация изменилась, некоторые браузеры поддерживали именно его. Чтобы обеспечить наилучшую совместимость, проверьте, существует ли свойство url. Если его не окажется, то выясните, определено ли свойство uri.

Событие `storage` невозможно отменить, и сама функция обратного вызова `handle_storage` никак не может предотвратить изменение. Браузер таким образом просто говорит нам: «Вот тут что-то случилось. Теперь уже ничего не поделаешь, но я решил вас поставить в известность».

Ограничения в современных браузерах

Говоря о прототипах локального хранилища (см. раздел «Краткая история прототипов локального хранилища до HTML5» этой главы), использующих сторонние приложения, я упомянул об ограничениях, свойственных каждому из этих механизмов: объем хранимых данных и т. п. Однако пока ничего не было сказано о функциональных ограничениях HTML5-хранилища, ставшего стандартным.

По умолчанию каждый домен имеет 5 Мбайт памяти на хранение данных. Удивительно, с каким единодушием браузеры сошлись на этой величине, которая в спецификации HTML5 всего лишь рекомендательна. Надо помнить, что хранимые данные имеют строковый тип, а не исходный. Эта разница в представлении становится явной, например при хранении большого количества целых чисел или десятичных дробей. Каждый разряд числа с плавающей запятой хранится не обычным для таких чисел образом, а как символ.

При превышении указанного объема хранимых данных будет возбуждено исключение `QUOTA_EXCEEDED_ERR`. На естественный вопрос, может ли браузер запросить у пользователя больше места для хранения данных, ответ будет отрицательным. На момент написания этой главы в браузерах не был реализован какой-либо механизм, позволяющий веб-программисту с согласия пользователя расширять область хранения. Некоторые браузеры (например, Opera) разрешают пользователю определять квоты хранилища для каждого сайта независимо. Однако такое возможно лишь по воле самого пользователя и не может быть инициировано каким-либо элементом веб-приложения.

HTML5-хранилище в действии

Рассмотрим пример практического использования HTML5-хранилища. Для этого вновь обратимся к игре «Уголки», которую мы реализовали в разделе «Живой

пример» главы 4. В нашей реализации есть одна проблема: если закрыть окно браузера в процессе, то текущее состояние и результаты будут утрачены. HTML5-хранилище позволяет сохранять игровой процесс локально — в самом браузере. Можете открыть демонстрационную страницу http://diveintohtml5.org/examples/localstorage-halma.html, сделать несколько ходов, закрыть вкладку браузера, а затем восстановить ее. Если в вашем браузере поддерживается HTML5-хранилище, то демостраница волшебным образом «вспомнит» состояние игры: сколько ходов вы сделали, как была расположена на доске каждая из фишек, была ли выбрана одна из фишек (если да, то какая именно).

Как работает этот механизм? Каждый раз при изменении состояния игры вызывается следующая функция:

```
function saveGameState() {
  if (!supportsLocalStorage()) { return false; }
  localStorage["halma.game.in.progress"] = gGameInProgress;
  for (var i = 0; i < kNumPieces; i++) {
    localStorage["halma.piece." + i + ".row"] = gPieces[i].row;
    localStorage["halma.piece." + i + ".column"] = gPieces[i].column;
  }
  localStorage["halma.selectedpiece"] = gSelectedPieceIndex;
  localStorage["halma.selectedpiecehasmoved"] = gSelectedPieceHasMoved;
  localStorage["halma.movecount"] = gMoveCount;
  return true;
}
```

Как можно видеть, объект localStorage «узнает», есть ли начатая и неоконченная игра (свойство gGameInProgress булевого типа). Если такая игра есть, то перебираются все фишки на игровом поле (gPieces, массив JavaScript) и в хранилище попадают строки и столбцы, на пересечении которых находятся фишки. После этого сохраняются некоторые дополнительные параметры: выбранная фишка (gSelectedPieceIndex, целое число), возможность продолжения серии переносов этой выбранной фишки (gSelectedPieceHasMoved булевого типа), общее количество сделанных ходов (gMoveCount, целое число).

При загрузке страницы мы не вызываем автоматически функцию newGame(), которая бы присвоила всем этим переменным значения по умолчанию, а вызываем resumeGame(). Опираясь на возможности HTML5-хранилища, функция resumeGame() проверяет, нет ли локально сохраненного состояния игры. Если есть, то значения будут восстановлены из объекта localStorage:

```
function resumeGame() {
  if (!supportsLocalStorage()) { return false; }
  gGameInProgress = (localStorage["halma.game.in.progress"] == "true");
  if (!gGameInProgress) { return false; }
  gPieces = new Array(kNumPieces);
  for (var i = 0; i < kNumPieces; i++) {
    var row = parseInt(localStorage["halma.piece." + i + ".row"]);
    var column = parseInt(localStorage["halma.piece." + i + ".column"]);
    gPieces[i] = new Cell(row, column);
  }
```

```
  gNumPieces = kNumPieces;
  gSelectedPieceIndex = parseInt(localStorage["halma.selectedpiece"]);
  gSelectedPieceHasMoved = localStorage["halma.selectedpiecehasmoved"] == "true";
  gMoveCount = parseInt(localStorage["halma.movecount"]);
  drawBoard();
  return true;
}
```

Для этой функции очень важна оговорка, которую я сделал выше и повторю здесь: *данные хранятся в виде строк; если вы хотите работать с данными не строкового типа, их надо конвертировать при получении*. Так, например, флаг «Идет игра» (gGameInProgress) имеет булев тип. В функции saveGameState() мы сохраняем его, не заботясь о типе данных:

```
localStorage["halma.game.in.progress"] = gGameInProgress;
```

Но в функции resumeGame() надо рассматривать значение, взятое из локального хранилища, как строку. Соответствующее логическое значение мы должны построить сами:

```
gGameInProgress = (localStorage["halma.game.in.progress"] == "true");
```

Так же и целое количество ходов (gMoveCount) в функции saveGameState() просто сохраняется:

```
localStorage["halma.movecount"] = gMoveCount;
```

В функции же resumeGame() мы должны привести значение к целочисленному типу, используя стандартную функцию JavaScript parseInt():

```
gMoveCount = parseInt(localStorage["halma.movecount"]);
```

Альтернативы: хранилище без ключей и значений

Хотя прошлое HTML5-хранилища изобилует уловками и обходными путями (см. раздел «Краткая история прототипов локального хранилища до HTML5» этой главы), его нынешнее состояние на удивление благополучно. Новый API был стандартизован и реализован во всех основных браузерах, на большинстве платформ и устройств. У веб-разработчика не каждый день такой праздник, верно? Но вообще современные опыты не ограничиваются пресловутыми 5 Мбайт и именованными парами «ключ — значение». Вот почему есть несколько альтернативных точек зрения на будущее постоянного локального хранилища.

Одна альтернатива уже знакома вам по крайней мере своим трехбуквенным названием: *SQL*. Разработанный специалистами Google в 2007 году кроссбраузерный плагин Gears с открытым исходным кодом имел встроенную базу данных на основе SQLite. Этот ранний прототип впоследствии повлиял на разработку спецификации Web SQL Database. База данных Web SQL (ранее известная как WebDB) — тонкая обертка вокруг базы данных SQL. С ее помощью можно из JavaScript делать, например, следующее:

```
openDatabase('documents', '1.0', 'Local document storage', 5*1024*1024,
  function (db) {
  db.changeVersion('', '1.0', function (t) {
    t.executeSql('CREATE TABLE docids (id, name)');
  }, error);
});
```

Как можно видеть, основное действие запрограммировано в строке с методом ExecuteSQL. Здесь могла бы быть любая другая корректная SQL-команда с использованием SELECT, UPDATE, INSERT или DELETE. Итак, серверное программирование баз данных теперь доступно прямо из JavaScript!

В табл. 7.3 показано, в каких браузерах реализована спецификация Web SQL Database.

Таблица 7.3. Поддержка Web SQL Database

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
–	–	4.0+	4.0+	10.5+	3.0+	–

Конечно, если вам доводилось пользоваться несколькими разными СУБД, то вы понимаете, что SQL — это скорее маркетинговый термин, а не жестко регламентированный стандарт (иногда то же самое говорят об HTML5, но это неважно). Есть действующая спецификация SQL, она называется SQL-92; но ни один сервер баз данных не соответствует ей и только ей. Есть SQL компании Oracle, SQL Microsoft, SQL в MySQL, SQL в PostgreSQL, SQL в SQLite. Более того, в каждом из этих продуктов функциональность SQL со временем расширяется, так что даже обозначения вида «SQL в SQLite» оказываются недостаточными. Надо говорить: «версия SQL, которая поставляется с SQLite версии X.Y.Z».

Все это подводит нас к следующему замечанию, с которого ныне начинается спецификация Web SQL:

> Эта спецификация в своем развитии достигла тупика. Все заинтересованные разработчики пользуются одной и той же серверной системой SQL (а именно, SQLite), но, чтобы двигаться по пути стандартизации, надо иметь несколько независимых реализаций. Пока не найдется разработчик, заинтересованный в реализации данной спецификации на основе другой СУБД, описание диалекта SQL здесь представляет собой просто ссылку на Sqlite. Для настоящего стандарта такое положение вещей неприемлемо.

Вот тот фон, на который следует проецировать мой рассказ о другой альтернативе. Это мощное локальное хранилище для веб-приложений Indexed Database API, ранее известное как WebSimpleDB. Теперь его кратко называют IndexedDB.

Indexed Database API — так называемое *хранилище объектов*. По своей идейной базе хранилища объектов родственны SQL-СУБД. Есть базы данных с записями, в каждой из которых определенное количество полей. Каждому полю присвоен тип данных, устанавливаемый при создании базы. Можно выбрать некоторое подмножество записей и перечислить их с помощью «курсора». Изменения в хранилище объектов происходят в рамках «транзакций».

Если вы знакомы с программированием баз данных на SQL, то все эти термины, вероятно, знакомы вам. Основная разница в том, что у хранилища объектов нет

собственного структурированного языка запросов. Нельзя построить команду вида `"SELECT * from USERS where ACTIVE = 'Y'"`. Вместо этого используются методы хранилища объектов, позволяющие открыть базу `USERS`, перечислить записи, отсеять записи, соответствующие неактивным пользователям, и востребовать значения всех полей в оставшихся записях с помощью методов-акцессоров. Статья «IndexedDB для начинающих» (http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/) — хорошее руководство по функциональности IndexedDB. В ней также скрупулезно сравниваются IndexedDB и Web SQL.

Ко времени написания этой главы система IndexedDB не была реализована ни в одном популярном браузере. В начале июня 2010 года разработчики Mozilla сообщили, что «в следующие несколько недель будет несколько тестовых сборок с IndexedDB», но до сих пор неизвестно, будет ли этот механизм включен в основной релиз Firefox 4. Зато фонд Mozilla заявил, что в их браузере не будет реализована спецификация Web SQL. Google рассматривает возможность поддержки IndexedDB в Chromium и Google Chrome. Даже Microsoft полагает, что IndexedDB — «отличное решение для веб-разработки».

Итак, чем для вас как для веб-разработчика полезна система IndexedDB? В данный момент абсолютно ничем. А через год? Возможно, как-то уже и пригодится. В разделе «Для дальнейшего изучения» есть ссылки на несколько хороших пособий начального уровня по IndexedDB.

Для дальнейшего изучения

HTML5-хранилище:

- спецификация HTML5-хранилища (http://dev.w3.org/html5/webstorage/);
- «Введение в DOM-хранилище» (http://msdn.microsoft.com/en-us/library/cc197062(VS.85).aspx) на сайте MSDN;
- «Веб-хранилище: простое и мощное хранилище данных на клиентской стороне» (http://dev.opera.com/articles/view/web-storage/), статья Шуэтанка Диксита (Shwetank Dixit);
- статья «DOM-хранилище» (https://developer.mozilla.org/en/dom/storage) на сайте Центра Mozilla для веб-разработчиков;



ПРИМЕЧАНИЕ

По большей части эта статья посвящена реализованному в Firefox прототипу — объекту globalStorage, нестандартному предшественнику объекта localStorage. Поддержка интерфейса localStorage появилась в Mozilla Firefox начиная с версии 3.5.

- «Локальное хранилище HTML5 для мобильных веб-приложений» (http://www.ibm.com/developerworks/xml/library/x-html5mobile2/), руководство на сайте IBM для разработчиков.

Предшествующие HTML5 разработки Брэда Ньюберга и соавторов:

- «В Internet Explorer есть встроенная поддержка постоянного хранения данных?!?!» (http://codinginparadise.org/weblog/2005/08/ajax-internet-explorer-has-native.html), об объекте `userData` в IE;

- хранилище Dojo (http://docs.google.com/View?docid=dhkhksk4_8gdp9gr#dojo_storage) — часть более обширного руководства по использованию библиотеки Dojo Offline (ныне вышедшей из употребления);
- справка по прикладному интерфейсу `dojox.storage.manager` (http://api.dojotoolkit.org/jsdoc/HEAD/dojox.storage.manager);
- SVN-репозиторий `dojox.storage` (http://svn.dojotoolkit.org/src/dojox/trunk/storage/).

Web SQL:

- спецификация Web SQL Database (http://dev.w3.org/html5/webdatabase/);
- «Знакомство с базами данных Web SQL» (http://html5doctor.com/introducing-web-sql-databases/) — статья Реми Шарпа;
- демонстрация работы Web Database (http://html5demos.com/database);
- сценарий `persistence.js` (http://zef.me/2774/persistence-js-an-asynchronous-javascript-orm-for-html5gears) — «асинхронная ORM на JavaScript», построенная на основе Web SQL и Gears.

IndexedDB:

- спецификация Indexed Database API (http://dev.w3.org/2006/webapi/IndexedDB/).
- «Не только HTML5: API баз данных и путь к IndexedDB» (http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/) — статья Эрана Ранганатана (Arun Ranganathan) и Шона Уилшера (Shawn Wilsher);
- «Firefox 4 и IndexedDB: курс молодого бойца» (http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/) — статья Эрана Ранганатана и Шона Уилшера.

8 На волю, в офлайн!

Приступим

Что такое офлайновое веб-приложение? На первый взгляд перед нами терминологическое противоречие. Веб-страница загружается, прежде чем отображаться, а загрузка требует подключения к Сети. Можно ли скачивать данные, находясь вне Сети? Нет, конечно. Но ведь *можно* скачивать заблаговременно, находясь в Сети. На этом принципе и основана система офлайновых приложений в HTML5.

В простейшем случае офлайновое веб-приложение представляет собой список адресов HTML-страниц, CSS-таблиц, файлов JavaScript, изображений и любых других ресурсов. На этот список — так называемый *манифест*, обычный текстовый файл, хранящийся где-либо на веб-сервере, — указывает главная страница офлайнового веб-приложения. Браузер, в котором реализована система офлайновых приложений HTML5, прочтет список URL-адресов из манифеста, скачает указанные там ресурсы, поместит их в локальный кэш и будет хранить эти локальные копии вплоть до момента их изменения. Когда в следующий раз вы попытаетесь запустить веб-приложение, не имея подключения к Сети, браузер автоматически переключится на локальную копию.

С этого момента начинается работа веб-программиста. В DOM есть свойство, которое показывает, в Сети находится пользователь или нет. С изменением значения этого свойства (например, сейчас вы работаете в автономном режиме, а через несколько секунд подключаетесь к Сети или наоборот) связаны определенные события. И это далеко не вся интересующая нас функциональность. Если приложение порождает какую-либо информацию или сохраняет свои состояния, то можно хранить все это локально (см. главу 7), пока пользователь находится вне Сети, и синхронизировать с удаленным сервером после подключения. Иными словами, HTML5 позволяет перенести приложение в офлайн, но как будет организована его автономная работа — решать веб-мастеру.

В табл. 8.1 показано, в каких браузерах поддерживаются офлайновые веб-приложения.

Таблица 8.1. Поддержка офлайна

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
–	√	√	√	–	√	√

Манифест кэша

Ключевой для офлайнового веб-приложения файл — манифест кэша. Как я уже говорил, это список всех ресурсов, которые ваше веб-приложение может востребовать, пока оно отключено от Сети. Чтобы запустить процесс загрузки и кэширования этих ресурсов, надо сослаться на файл манифеста в атрибуте `manifest` тега `<html>`.

```
<!DOCTYPE HTML>
<html manifest="/cache.manifest">
  <body>
    …
  </body>
</html>
```

Файл манифеста кэша может располагаться где угодно на веб-сервере, но он должен быть обязательно оснащен MIME-типом `text/cache-manifest`. Если вы работаете с веб-сервером на основе Apache, то стоит добавить соответствующую директиву `AddType` в файл с расширением `.htaccess` в корневой директории:

```
AddType text/cache-manifest .manifest
```

После этого убедитесь, что имя вашего файла манифеста заканчивается на `.manifest`. Если вы пользуетесь иным веб-сервером или нестандартной конфигурацией Apache, обратитесь к справочной документации вашего сервера по вопросу об управлении заголовками `Content-Type`.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Мое веб-приложение состоит из нескольких страниц. Нужно ли указывать атрибут manifest на каждой странице или достаточно поместить его только на главной странице?

Ответ: Каждая страница вашего веб-приложения должна ссылаться посредством атрибута manifest на файл — манифест кэша всего веб-приложения.

Допустим, что формальности выполнены: каждая из ваших HTML-страниц указывает на файл манифеста кэша, а сам файл обслуживается правильным заголовком `Content-Type`. Теперь разберемся, что *представляет собой* файл манифеста.

Первая строка файла манифеста обязательно такова:

```
CACHE MANIFEST
```

Последующее содержимое файла манифеста делится на три раздела: явное (explicit), резервное (fallback) и сетевое (NETWORK, или online whitelist — «онлайновый белый список»). Заголовок каждого раздела оформляется отдельной строкой. Если никаких заголовков в файле манифеста нет, то подразумевается, что все перечисленные в нем ресурсы — явные. Чтобы не разболелась голова, постарайтесь пока не обращать внимания на терминологию.

Вот образец правильно построенного файла манифеста. В нем перечислены три ресурса: CSS-файл, файл JavaScript и изображение в формате JPEG:

```
CACHE MANIFEST
/clock.css
/clock.js
/clock-face.jpg
```

Этот файл манифеста кэша не содержит заголовков. Значит, все перечисленные в нем ресурсы по умолчанию явные, то есть браузер скачивает и локально кэширует их, чтобы использовать вместо соответствующих онлайновых файлов при отключении от Сети. Таким образом, после загрузки показанного выше файла манифеста из корневой директории веб-сервера будут скачаны `clock.css`, `clock.js` и `clock-face.jpg`. Если отсоединить сетевой кабель и обновить страницу, то все эти ресурсы будут работать по-прежнему.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Нужно ли перечислять в манифесте кэша HTML-страницы?

Ответ: Единого ответа на этот вопрос не существует. Если ваше веб-приложение состоит из одной страницы, просто убедитесь, что она ссылается на манифест кэша посредством атрибута manifest. Всегда, когда вы переходите на HTML-страницу с атрибутом manifest, сама эта страница выступает как часть веб-приложения, так что указывать ее в файле манифеста не нужно. Однако, если ваше веб-приложение состоит из нескольких страниц, надо перечислить их все в файле манифеста. В ином случае браузер не будет знать, что есть и другие HTML-страницы (кроме основной), которые следует загрузить и кэшировать.

Раздел NETWORK

Рассмотрим чуть более сложный пример. Например, вы хотите, чтобы приложение-часы отслеживало посещения страницы с помощью сценария `tracking.cgi`, динамически загружаемого из атрибута `<img src>`. Если кэшировать этот ресурс, то следить за посетителями не удастся. Значит, этот ресурс никогда не должен быть доступен в автономном режиме. Вот как добиться поставленной цели:

```
CACHE MANIFEST
NETWORK:
/tracking.cgi
CACHE:
/clock.css
/clock.js
/clock-face.jp
```

Этот файл манифеста содержит *заголовки разделов*. Со строки `NETWORK:` начинается сетевой раздел — онлайновый белый список. Ресурсы в нем никогда не кэшируются и недоступны автономно (попытка загрузить один из таких ресурсов в офлайновом режиме приведет к ошибке). Со строки `CACHE:` начинается явный раздел, содержимое которого совпадает с предыдущим примером. Каждый из трех перечисленных здесь ресурсов будет кэширован и доступен в режиме офлайн.

Раздел FALLBACK

В файле манифеста кэша есть еще *резервный раздел*. В нем можно перечислить заместители тех интернет-ресурсов, которые по тем или иным причинам не были (или вообще не могут быть) сохранены в кэше. В спецификации HTML5 приводится следующий интересный вариант использования «резервного» раздела:

```
CACHE MANIFEST
FALLBACK:
/ /offline.html
NETWORK:
*
```

Как работает эта запись? Рассмотрим сайт с миллионами страниц вроде «Википедии». По-видимому, загрузить такой сайт целиком не представляется возможным, да и незачем. Но предположим, что вы можете сделать *часть* страниц доступными в офлайне. Как решить, какие из страниц кэшировать? Возможен такой механизм: каждая просмотренная пользователем страница «Википедии», гипотетически доступной в офлайне, скачивается и кэшируется. Значит, кэш будет содержать все статьи, которые когда-либо открывались в браузере, все страницы обсуждения (где пользователь может беседовать с другими участниками по поводу содержания статей) и страницы правок (где пользователь может вносить изменения в статьи).

Этот механизм и призван реализовать манифест кэша. Пусть каждая HTML-страница нашей «Википедии» (энциклопедическая статья, страница обсуждения, правок или истории) ссылается на этот файл манифеста. При посещении любой подобной страницы браузер говорит: «Так-так, эта страница — часть офлайнового приложения. Что мне уже известно об этом приложении?» Если браузер еще ни разу не скачивал соответствующий файл манифеста, то будет создан новый офлайновый кэш приложения, программа скачает все ресурсы, перечисленные в манифесте кэша, и добавит текущую страницу в кэш приложения. А вот если браузер «знает» о данном манифесте кэша, то он просто добавит текущую страницу в кэш приложения, который уже существует. Так или иначе, каждая посещенная страница попадает в кэш приложения. Это важно: оказывается, можно сделать офлайновое веб-приложение, которое бы «лениво» добавляло страницы по мере их посещения. Значит, перечислять все ваши HTML-страницы в манифесте кэша не обязательно.

Теперь посмотрим на резервный раздел `FALLBACK:`, который в данном манифесте кэша состоит из одной строки. Часть строки до пробела — это в действительности не URL-адрес, а *шаблон* URL-адреса. Одиночный символ слеша (`/`) соответствует любой странице вашего сайта, а не только главной. При попытке открыть страницу в автономном режиме браузер попробует найти ее в кэше приложения. Если страница найдется (потому что пользователь посещал ее, пока работал в Сети, и в это время страница была автоматически внесена в кэш приложения), то в браузере отобразится кэшированная копия. Если же страница не найдется, то вместо сообщения об ошибке браузер выведет на экран страницу `/offline.html`, как и указано во второй половине резервной строки.

Наконец, рассмотрим сетевой раздел NETWORK:. В показанном выше манифесте кэша он также занимает одну строку, которая состоит из единственного символа (*). В сетевом разделе этот символ имеет особое значение и называется «подстановочный символ онлайнового белого списка». С его помощью мы говорим: все, что отсутствует в кэше приложения, должно быть скачано с исходного адреса, если доступно подключение к Интернету. Для «открытых» офлайновых веб-приложений это может быть важно. Если вернуться к нашему примеру, то содержательно получится следующее: пока пользователь, работая в Сети, просматривает нашу «Википедию» с возможностью автономной работы, его браузер будет скачивать изображения, видео и другие встроенные ресурсы обыкновенным образом, даже если они принадлежат другому домену. Это вообще свойственно большим сайтам, даже тем, которые не составляют часть офлайнового веб-приложения: HTML-страницы генерируются и обслуживаются локально, а картинки и видеоклипы загружаются через сетевую платформу распределения контента (CDN) на другом домене.

Без подстановочного символа гипотетическая «Википедия» с поддержкой офлайна будет вести себя неожиданно: в частности, при работе в режиме онлайн она не будет загружать изображения и видео с внешних серверов.

Полон ли наш пример? Нет. «Википедия» — это не только HTML-файлы. В ней на каждой странице используются распределенные файлы CSS, JavaScript и картинки. Все эти ресурсы следовало бы перечислять в явном виде в разделе CACHE: файла манифеста, чтобы при автономной работе страницы правильно отображались и отвечали на действия пользователя. Но весь смысл резервного раздела в том, чтобы сделать ваше офлайновое приложение «открытым», то есть способным выходить за рамки ресурсов, явным образом названных в файле манифеста.

Поток событий

Итак, об офлайновых веб-приложениях, манифесте кэша и кэше приложения я рассказал, но весьма расплывчато, в терминах, далеких от технических подробностей. Что-то загружается, браузеры принимают какие-то решения, вот все и работает. Но мы ведь говорим о веб-разработке, а в ней ничего не работает так просто, без всяких усилий.

Сейчас поговорим о потоке событий, а именно о DOM-событиях. Когда браузер открывает страницу, которая ссылается на манифест кэша, в объекте window.applicationCache последовательно срабатывает несколько событий. Понимаю, что рассказ об этом будет выглядеть сложновато, но, поверьте, изложить все проще и без потери важных подробностей не представляется мне возможным.

Вот какова процедура.

1. Как только браузер усматривает атрибут manifest в теге <html>, происходит событие checking объекта window.applicationCache (все события, о которых здесь и далее идет речь, принадлежат объекту window.applicationCache). Событие checking срабатывает всегда, вне зависимости от того, посещал ли ранее пользователь эту страницу или любую другую, которая ссылается на тот же манифест кэша.

2. Если данный манифест кэша обнаружен впервые:
   - происходит событие downloading, после которого браузер начинает загружать ресурсы, перечисленные в манифесте кэша;
   - пока идет загрузка, в браузере периодически срабатывает событие progress с информацией о том, сколько файлов уже скачано к текущему моменту и сколько осталось в очереди на загрузку;
   - после того как все ресурсы, названные в манифесте, окажутся благополучно загружены, в браузере происходит событие cached, которое заключает весь процесс; это сигнал, означающий, что офлайновое веб-приложение полностью кэшировано и готово к работе в автономном режиме.

   Вот все и готово.
3. С другой стороны, если пользователь уже посещал данную страницу или любую другую, которая ссылается на тот же манифест кэша, то браузер уже «знает» об этом манифесте и, возможно, содержит в кэше приложения часть названных там ресурсов или даже все приложение в работающем состоянии. Теперь вопрос в том, не изменился ли кэш с того момента, когда браузер обрабатывал его в последний раз.
   - Если ответ отрицательный, то есть манифест кэша остался прежним, то в браузере немедленно сработает событие noupdate — и все.
   - Если ответ положительный, то есть манифест кэша изменился, то в браузере произойдет событие downloading и каждый ресурс из числа указанных в манифесте будет загружаться повторно.

   Пока идет загрузка, в браузере будет периодически срабатывать событие progress с информацией о том, сколько файлов уже скачано к текущему моменту и сколько осталось в очереди на загрузку.

   После того как все ресурсы, названные в манифесте, окажутся благополучно загружены, в браузере произойдет событие updateready, которое заключает весь процесс. Это сигнал, означающий, что новая версия офлайнового веб-приложения полностью кэширована и готова к работе в автономном режиме. *Однако новая версия пока не используется.* Чтобы пользователю не пришлось перезагружать страницу и смена версии со старой на новую произошла автоматически, можно вызвать функцию window.applicationCache.swapCache().

Если во всем этом процессе что-нибудь однажды нарушится, в браузере сработает событие error и процедура остановится. Весьма урезанный список причин, ведущих к неполадкам, таков:

- манифест кэша возвращает HTTP-ошибку 404 (Страница не найдена) или 410 (Страница удалена);
- манифест кэша обнаружен и не изменился, но HTML-страницу, которая сослалась на манифест, оказалось невозможно загрузить;
- манифест кэша обнаружен и изменился, но браузер не сумел загрузить один из ресурсов, перечисленных в нем.

«Убей меня поскорее!», или Отладка как одно из изящных искусств

Хочу указать здесь на два важных момента. О первом только что шла речь, но вы, вероятно, не прониклись этой мыслью в достаточной мере. Поэтому повторяю: *если хотя бы один ресурс, упомянутый в файле манифеста, не будет скачан надлежащим образом, то весь процесс кэширования вашего офлайнового приложения закончится этой ошибкой*. Событие `error` в браузере не дает понять, что стало причиной ошибки. Поэтому отладка офлайнового веб-приложения — очень тягостное занятие.

Второй важный момент связан, если говорить технически, не с ошибкой, но склонен принимать вид серьезного сбоя в браузере; все будет выглядеть именно так, по крайней мере до тех пор, пока вы не поймете, что происходит. Речь идет о том, каким именно образом браузер проверяет, изменился ли файл манифеста кэша. В этом процессе три стадии. Сейчас будут скучные, но существенные детали, приготовьтесь к ним.

1. Исходя из обычной HTTP-семантики, браузер проверяет, истек ли срок действия манифеста кэша. Метаинформацию об этом файле, как и о любом другом файле, пересылаемом по HTTP, веб-сервер обычно включает в HTTP-заголовок ответа. Есть такие HTTP-заголовки (`Expires` и `Cache-Control`), которые сообщают браузеру, что файл разрешено кэшировать, не обращаясь к серверу за подтверждением. Этот вид кэширования никак не связан с офлайновыми веб-приложениями; ему подвергаются почти все HTML-страницы, таблицы стилей, сценарии, изображения и прочие сетевые ресурсы.
2. Если срок годности манифеста кэша, согласно его HTTP-заголовкам, истек, то браузер запрашивает сервер, появилась ли новая версия файла. Если да, то новая версия будет загружена. Для этого браузер создаст HTTP-запрос с той датой последнего изменения файла манифеста кэша, которая известна клиентской программе по HTTP-заголовку, сопровождавшему файл при его последнем скачивании. Если веб-сервер определит, что после этой даты файл манифеста не изменился, будет возвращен код состояния 304 (Не изменено). Все это также неспецифично для офлайновых веб-приложений, а происходит фактически со всеми сетевыми ресурсами.
3. Если веб-сервер полагает, что файл манифеста изменился с даты, известной клиенту, то будет возвращен код HTTP-состояния 200 (OK), а вслед за этим сервер отправит клиенту содержимое нового файла с новыми заголовками `Cache-Control` и соответственно с новой датой изменения, что обеспечит корректное выполнение этапов 1 и 2 в следующий раз. (Вот в чем мощь HTTP! Веб-серверы всегда заглядывают в будущее. Обязанный отправить вам файл, веб-сервер будет, однако, всеми доступными ему способами избегать немотивированной повторной пересылки.) Как только новый файл манифеста кэша будет загружен, браузер сопоставит его содержимое с уже известным экземпляром. Если содержимое осталось таким же, каким оно было прежде, то браузер не станет повторно загружать ни один из ресурсов, перечисленных в манифесте.

В ходе разработки и тестирования вашего собственного офлайнового веб-приложения для вас может оказаться проблематичным каждый из этих трех шагов. Например, вы разместили на сервере какой-либо вариант файла манифеста кэша, но через десять минут вдруг стало ясно, что следует добавить еще один ресурс. Разве это проблема? Добавим одну строку и перезальем файл. Увы. Вот вы обновили страницу, браузер нашел атрибут `manifest`, произошло событие `checking`, и... дальнейшее молчание. Ваш браузер будет упрямо настаивать на том, что файл манифеста кэша не изменился. Дело, скорее всего, в следующем: ваш веб-сервер по умолчанию сконфигурирован так, чтобы предлагать браузерам кэшировать статические файлы в течение нескольких часов (используется HTTP-семантика, а именно заголовки `Cache-Control`). Значит, браузер так и не преодолеет первую стадию процедуры проверки. Веб-серверу, конечно, известно, что файл подвергся изменениям, но браузер даже не предполагает, что веб-сервер можно об этом спросить. В чем же дело? При последнем скачивании манифеста кэша веб-сервер велел браузеру (посредством заголовков `Cache-Control`) держать ресурсы в кэше в течение нескольких часов. Что, в сущности, и происходит сейчас — спустя десять минут.

Строго говоря, это не ошибка. Все работает именно так, как должно. Если бы веб-серверы не могли принуждать браузеры (и промежуточные прокси-серверы) кэшировать данные, Сеть обрушилась бы в считаные часы. Но это слабое утешение для того, кто потратил множество времени на тщетные попытки выяснить, почему же браузер не видит обновленный манифест кэша. (И даже больше: если вы возились достаточно долго, все таинственным образом заработает снова! Ведь срок хранения данных в кэше истечет, как и следует ожидать!)

Чтобы избавиться от головной боли, вам совершенно точно надо будет сделать следующее: сконфигурировать веб-сервер так, чтобы файл манифеста перестал быть кэшируемым с помощью семантики HTTP. Если вы работаете с веб-сервером на основе Apache, то достаточно будет следующих двух строк в файле `.htaccess`:

```
ExpiresActive On
ExpiresDefault "access"
```

Это способ запретить кэширование всех файлов в данной директории и всех ее поддиректориях. Вероятно, на деле вам придется как-то дополнить это решение. Надо либо прибавить директиву `<Files>`, чтобы отключить кэширование только файла манифеста, либо создать поддиректорию, которая содержала бы только файл `.htaccess` и файл манифеста. Технические детали конфигурирования, как и всегда, для разных веб-серверов различны, поэтому не помешает обратиться к справочной документации о том, как управлять HTTP-заголовками кэширования.

Отключить HTTP-кэширование файла манифеста не значит решить разом все проблемы. Будут еще и ситуации, когда один из ресурсов, хранимых в кэше приложения, изменился, но его URL-адрес на веб-сервере остался прежним. Здесь неполадка возникает на второй стадии процесса. Если не изменился сам файл манифеста, то браузер так и не заметит, что был изменен один из ресурсов, кэшированных ранее.

Рассмотрим следующий пример:

```
CACHE MANIFEST
# rev 42
clock.js
clock.css
```

Изменив и перезалив таблицу стилей `clock.css`, вы не увидите разницы: файл манифеста не поменяется. Поэтому каждый раз, внося изменения в один из ресурсов вашего офлайнового веб-приложения, обязательно меняйте и сам файл манифеста. Достаточно такой несложной вещи, как замена одного символа. Насколько мне известно, проще всего включать в файл манифеста комментарий с порядковым номером версии. Когда меняется один из ресурсов, меняйте версионный номер в комментарии. Веб-сервер передаст браузеру новый, измененный файл манифеста, а браузер, видя, что содержимое файла изменилось, повторно скачает все ресурсы, перечисленные в манифесте:

```
CACHE MANIFEST
# rev 43
clock.js
clock.css
```

Строим офлайновое приложение

Помните игру «Уголки», которая впервые появилась в этой книге в разделе «Живой пример» главы 4 и была улучшена, когда мы стали сохранять состояния игры в локальном хранилище (см. раздел «HTML5-хранилище в действии» главы 7)? Выпустим «Уголки» на волю — в офлайн. Для этого надо составить список всех ресурсов, от которых зависит игра. Есть главная HTML-страница, есть один файл JavaScript со всем кодом игры — и все! Картинок нет, потому что прорисовка выполняется программно с помощью API холста, а все необходимые CSS-стили содержатся в теге `<style>` в верхней части страницы. Таким образом, манифест кэша выглядит так:

```
CACHE MANIFEST
halma.html
../halma-localstorage.js
```

Пару слов о путях к файлам. В директории examples/ я создал поддиректорию offline/, куда и поместил файл манифеста кэша. Поскольку HTML-страницы нуждаются в одном маленьком дополнении, чтобы работать офлайн (об этом ниже), была создана отдельная копия HTML-файла, которую я тоже поместил в папку offline/. Но ввиду того, что код на JavaScript остался прежним с того времени, как в игру вступило локальное хранилище (см. раздел «HTML5-хранилище в действии» главы 7), я повторно использую тот самый файл с расширением JS, хранящийся в родительской папке examples/. В совокупности размещение всех файлов таково:

```
/examples/localstorage-halma.html
/examples/halma-localstorage.js
/examples/offline/halma.manifest
/examples/offline/halma.html
```

В файле манифеста (`/examples/offline/halma.manifest`) мы хотим сослаться на два файла: во-первых, на офлайновую версию HTML-файла (`/examples/offline/halma.html`), путь к которой описан в файле манифеста без префикса, и, во-вторых, на файл JavaScript из родительской папки (`/examples/halma-localstorage.js`). Путь к нему в файле манифеста относительный: `../halma-localstorage.js`. Аналогичным образом можно использовать относительные пути в атрибуте `src` тега `<img>`. Как будет ясно из следующего примера, допускается также использование абсолютных путей

(то есть таких, которые начинаются с корня текущего домена) и даже абсолютных URL-адресов (которые указывают на ресурсы, размещенные на других доменах).

Теперь мы должны добавить в HTML-файл атрибут manifest, ссылающийся на файл манифеста кэша:

```
<!DOCTYPE html>
<html lang="en" manifest="halma.manifest">
```

Вот и все! Когда браузер с поддержкой офлайновых приложений впервые загрузит нашу HTML-страницу, он скачает связанный с ней файл манифеста и начнет сохранять все перечисленные в нем ресурсы в кэше приложения, загружая их. С этого времени при повторных посещениях страницы в Сети офлайновый алгоритм будет перезагружаться. Поскольку в «Уголки» стало возможно играть автономно, а позиции незаконченных игр локально сохраняются, то пользователь может сколь угодно часто прерывать партию и возвращаться к ней.

Для дальнейшего изучения

Стандарты: «Офлайновые веб-приложения в спецификации HTML5» (http://bit.ly/cCkWZa).

Документация от разработчиков браузеров:

- «Офлайновые ресурсы в Firefox» (https://developer.mozilla.org/En/Offline_resources_in_Firefox) на сайте Центра Mozilla для веб-разработчиков;
- «Кэш офлайновых приложений HTML5» (http://developer.apple.com/safari/library/documentation/iPhone/Conceptual/SafariJSDatabaseGuide/OfflineApplicationCache/OfflineApplicationCache.html) — часть «Руководства по программированию для локального хранилища и офлайновых приложений в среде Safari» (http://developer.apple.com/safari/library/documentation/iPhone/Conceptual/SafariJSDatabaseGuide/Introduction/Introduction.html).

Учебные и демонстрационные материалы:

- «Об использовании HTML5 в платформе Gmail для мобильных устройств. Автономная работа с помощью кэша приложения. Часть 1» (http://googlecode.blogspot.com/2009/04/gmail-for-mobile-html5-series-using.html) — статья Эндрю Грива (Andrew Grieve);
- «Об использовании HTML5 в платформе Gmail для мобильных устройств. Автономная работа с помощью кэша приложения. Часть 2» (http://googlecode.blogspot.com/2009/05/gmail-for-mobile-html5-series-part-2.html) — статья Эндрю Грива;
- «Об использовании HTML5 в платформе Gmail для мобильных устройств. Автономная работа с помощью кэша приложения. Часть 3» (http://googlecode.blogspot.com/2009/05/gmail-for-mobile-html5-series-part-3.html) — статья Эндрю Грива;
- «Отладка кэша офлайновых приложений HTML5» (http://jonathanstark.com/blog/2009/09/27/debugging-html-5-offline-application-cache/) — статья Джонатана Старка (Jonathan Stark);
- «Офлайновое HTML5-приложение для редактирования и публикации изображений» (http://hacks.mozilla.org/2010/02/an-html5-offline-image-editor-and-uploader-application/) — статья Пола Руже (Paul Rouget).

9 Веб-формы как форма безумия

Приступим

Вы знаете о веб-формах все, не так ли? Действительно, чего там сложного: создать контейнер `<form>`, добавить несколько полей `<input type="text">`, возможно, одно `<input type="password">` и увенчать конструкцию великолепной кнопкой `<input type="submit">`.

Увы, это лишь незначительная часть всех современных функций. В HTML5 появилось больше десятка новых типов полей ввода, которые теперь можно использовать в формах. Когда я говорю «использовать», я имею в виду «использовать прямо сейчас» — без каких-либо уловок или обходных путей. Не стоит обольщаться: я не хочу сказать, что все эти новые чудесные функции уже поддерживаются во всех браузерах. Отнюдь. В современных браузерах формы с новыми элементами по-настоящему в фаворитах, а вот в старых браузерах такие формы выглядят более прохладно. Но в каждом старом браузере, даже в IE6, благополучно поддерживается какой-то рудимент новой функциональности.

Подсказывающий текст

Первое, чем обогащены HTML5-формы по сравнению с прежним поколением веб-форм, — это подсказывающий текст в полях ввода. Подсказывающий текст отображается внутри поля до тех пор, пока оно пусто и не несет фокуса. Как только пользователь щелкнет на нем или перейдет к нему с помощью табулятора, подсказывающий текст исчезнет.

Вам, наверное, уже приходилось видеть подсказывающий текст. Так, в адресной строке Mozilla Firefox есть подсказывающий текст вида Поиск в закладках и журнале (рис. 9.1).



Рис. 9.1. Подсказывающий текст в строке поиска браузера Firefox

Если щелкнуть на адресной строке или перейти к ней с помощью табулятора, подсказывающий текст исчезнет (рис. 9.2).

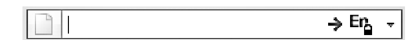


Рис. 9.2. Подсказывающий текст исчезает при наведении фокуса

Забавно, что Firefox 3.5 не поддерживает подсказывающий текст в веб-формах на страницах. Что ж, такова жизнь. В каких версиях браузеров работают подсказки, можно узнать из табл. 9.1.

Таблица 9.1. Поддержка подсказывающего текста

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
–	3.7+	4.0+	4.0+	–	–	–

С помощью следующего кода вы можете добавить подсказывающий текст в свои веб-формы:

```
<form>
  <input name="q" placeholder="Поиск в закладках и журнале">
  <input type="submit" value="Поиск">
</form>
```

Браузеры без поддержки атрибута `placeholder` его вполне безвредно игнорируют.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Можно ли пользоваться HTML-разметкой в атрибуте placeholder? А вдруг я захочу вставить картинку или, скажем, поменять цветовую схему?

Ответ: Атрибут placeholder — сугубо текстовый и не может содержать HTML-кода. Однако в некоторых браузерах стиль текста подсказок позволяет установить особые расширения CSS (http://trac.webkit.org/export/37527/trunk/LayoutTests/fast/forms/placeholder-pseudo-style.html).

Поля с автофокусировкой

На многих сайтах с помощью JavaScript реализована автоматическая фокусировка первого из полей веб-формы. Так, на главной странице Google.com форма поиска несет фокус, так что запрос можно вводить сразу, не помещая курсор в поле. Это удобно для большинства посетителей, но может мешать «продвинутым» пользователям и людям с особыми потребностями. Если на такой странице нажать Пробел, то прокрутка на один экран вниз не сработает, как можно ожидать; вместо этого в поле ввода появится пробел. Еще пример: если переместить фокус в другое из полей формы, пока страница грузится, то «услужливая» система вернет курсор обратно, в начальное поле. Это сбивает с ритма, к тому же пользовательский ввод попадает не туда, куда следует.

Все эти специальные случаи трудно учесть при автофокусировке с помощью JavaScript. Еще недавно пользователей, которых раздражает «кража» фокуса веб-страницами, ничем нельзя было утешить.

Для решения проблемы в HTML5 введен атрибут autofocus, который может быть применен к любому элементу веб-формы. Этот атрибут работает в полном соответствии с названием, то есть перемещает курсор в одно из полей ввода на форме. Но это не сценарное решение, а HTML-код, значит, его поведение на всех сайтах будет одинаково. Со своей стороны, разработчики браузеров и браузерных расширений, возможно, предложат пользователям функцию отключения автофокусировки.

В табл. 9.2 показано, какие из браузеров поддерживают автофокусировку.

Таблица 9.2. Поддержка автофокуса

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
–	–	4.0+	3.0+	10.0+	√	√

Вот как можно назначить автофокусировку одного из полей формы:

```
<form>
  <input name="q" autofocus>
  <input type="submit" value="Поиск">
</form>
```

Браузеры, которые не поддерживают атрибут autofocus, просто игнорируют его.

А если мы захотим, чтобы автофокусировка работала везде, а не только в HTML5-браузерах? Тогда придется остаться при нынешнем сценарии автофокусировки, сделав в нем два небольших изменения.

1. Добавить атрибут autofocus в HTML-код.
2. Протестировать поддержку атрибута autofocus в браузере (см. раздел «Автофокусировка в формах» главы 2). Запускать сценарий следует только в том случае, если встроенной поддержки атрибута autofocus в браузере нет.

```
<form name="f">

  <input id="q" autofocus>
  <script>
    if (!(“autofocus” in document.createElement(“input”))) {
      document.getElementById("q").focus();
    }
  </script>
  <input type="submit" value="Поехали">
</form>
...
```

Страница http://diveintohtml5.org/examples/input-autofocus-with-fallback.html являет собой пример автофокусировки с запасным (сценарным) вариантом.

СЕКРЕТЫ РАЗМЕТКИ

Многие веб-страницы для установки фокуса дожидаются события window.onload. Но оно не сработает, пока не будут загружены все картинки. Таким образом, если на странице много картинок, «наивный»

сценарий потенциально способен перенести фокус в одно из полей уже после того, как пользователь начал работать с другим элементом страницы (вот почему «продвинутые» пользователи так не любят сценарии автофокусировки). В предыдущем примере сценарий расположен в коде страницы непосредственно после поля формы, на которое он ссылается. Может случиться так, что ваша серверная система проявит недостаточно гибкости для реализации подобного решения. Тогда, если сценарий нельзя вставить посередине страницы, следует устанавливать фокус по срабатыванию такого пользовательского события, как $(document).ready() в jQuery, а не стандартного window.onload.

Адреса электронной почты

Больше десятка лет веб-формы могли включать поля лишь нескольких типов, самые популярные из которых перечислены в табл. 9.3.

Таблица 9.3. Типы полей ввода в HTML 4

Тип поля	**HTML-код**	**Примечание**
Флажок	<input type="checkbox">	Может быть установлен и снят
Переключатель	<input type="radio">	Способен объединяться в группы с другими полями ввода
Поле ввода пароля	<input type="password">	Вместо любых символов, вводимых пользователем, отображает точки
Раскрывающийся список	<select><option>…	
Выборщик файла	<input type="file">	Выводит окно открытия файла
Кнопка отсылки формы	<input type="submit">	
Поле текстового ввода	<input type="text">	Атрибут type может быть опущен

Все эти типы полей работают и в HTML5. При обновлении до HTML5 (то есть, как правило, при смене определения типа документа, о чем читайте в разделе «Определение типа документа» главы 3) вам не придется вносить в веб-формы никаких изменений. Да здравствует обратная совместимость!

Впрочем, в HTML5 появились некоторые новые типы полей, за использование которых (по причинам, которые сейчас станут ясны) единогласно выступают профессионалы.

Первый из этих новых типов полей предназначен для ввода адресов электронной почты. Вот код простейшей формы с таким полем:

```
<form>
  <input type="email">
  <input type="submit" value="Go">
</form>
```

Только что я с большим трудом удержался от того, чтобы написать: «В браузерах, которые не поддерживают `type="email"`…» Что же меня останавливает? Видите ли, я не уверен в смысле выражения «не поддерживать `type="email"`». В определенном смысле «поддерживают» `type="email"` все браузеры. Браузер может не

делать с таким полем ничего особенного (далее будет несколько примеров специфической обработки), но браузеры, незнакомые с `type="email"`, во всяком случае распознают такое поле как `type="text"` и отобразят его как обычное поле текстового ввода.

Мне буквально не хватает слов для доказательства важности этого факта. Миллионы форм в Интернете, в которых предлагается ввести адрес электронной почты, построены на основе `<input type="text">`. Пользователь видит перед собой поле текстового ввода, вписывает в это поле свой электронный адрес — и все. Но вот появляется HTML5, в котором определен `type="email"`. Какой будет реакция браузеров? Неужели они сойдут с ума? Вот и нет. Все браузеры мира, даже IE6, воспринимают неизвестное значение атрибута `type` как `type="text"`. Так что обновить веб-формы, вписывая в них `type="email"`, можно уже сейчас.

Что же имеется в виду, когда говорят, что браузер поддерживает `type="email"`? Собственно, что угодно. В спецификации HTML5 не предписан какой-либо определенный вид пользовательского интерфейса для новых типов полей. Opera помещает рядом с полем `type="email"` значок электронной почты. Другие HTML5-браузеры, например Safari и Chrome, отображают обычное поле текстового ввода, такое же, как `type="text"`. Поэтому пользователь, если только он не заглянет в исходный код страницы, так и не увидит разницы.

Есть еще iPhone. Физическая клавиатура у iPhone отсутствует, а текст набирается посредством прикосновений к экранной клавиатуре, которая всплывает в подходящее для этого время, например при фокусировке одного из полей формы на веб-странице. Браузер iPhone устроен очень интересно: распознавая некоторые HTML5-типы полей ввода, он *динамически меняет вид экранной клавиатуры*, чтобы соответствующие символы было удобнее вводить.

Так, например, адрес электронной почты — это текст, не правда ли? Да, но текст особого рода. Практически во всех адресах электронной почты содержатся символ `@` и по крайней мере одна точка (`.`), а пробела ожидать не следует. Поэтому, когда пользователь iPhone фокусирует на странице тег `<input type="email">`, всплывающая экранная клавиатура содержит пробел меньше обычных размеров, а также специальные клавиши для символов `.` и `@`, как показано на рис. 9.3.

Если подвести итог, то окажется, что нет никаких препятствий, которые бы заставляли повременить с превращением полей ввода электронных адресов в `type="email"`. Этого превращения, правда, почти никто не заметит, кроме пользователей iPhone. Но кто заметит, тот, тихонько улыбнувшись, поблагодарит вас за возможность работать в Сети чуть-чуть проще.



Рис. 9.3. Клавиатура, оптимизированная для ввода адреса электронной почты

Веб-адреса

Веб-адреса, известные многим как URL, а малочисленным педантам как URI, — это еще один тип специализированного текста. Соответствующие сетевые стандарты определяют синтаксис веб-адресов. Если веб-приложение просит вас ввести в форму веб-адрес, то ожидается, что вы напишете что-то наподобие http://www.google.com/, а не «Петровка, 38». В адресах часто применяются правые слеши и точки, а пробелы запрещены. Каждому веб-адресу свойствен суффикс домена: .com, .org и др.

Леди и джентльмены, представляю вашему вниманию (барабанная дробь, все глаза прикованы к сцене)... `<input type="url">`! На iPhone этот тип поля текстового ввода выглядит, как показано на рис. 9.4.

Как и в случае с адресом электронной почты, iPhone предоставляет пользователю виртуальную клавиатуру особого вида, приспособленную для веб-адресов. Пробел здесь полностью заменен тремя виртуальными клавишами: правый слеш (/), точка (.) и .com. Удерживая нажатой клавишу .com, можно выбрать другой популярный домен: .org, .net и т. п.

Браузеры без поддержки HTML5 будут обрабатывать `type="url"` в точности так же, как `type="text"`, что снимает все вопросы по поводу использования полей этого типа для ввода веб-адресов.

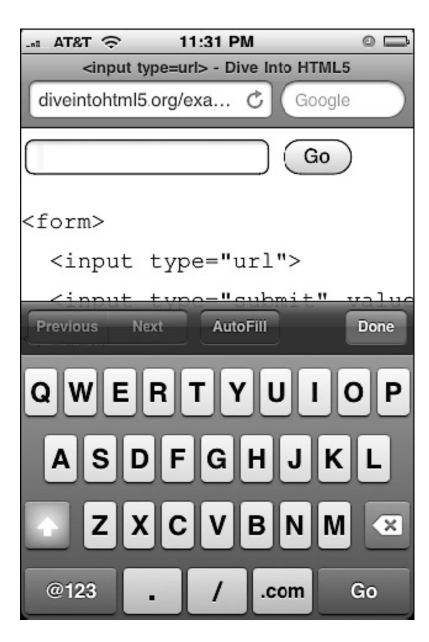


Рис. 9.4. Клавиатура, оптимизированная для ввода URL-адреса

Числа как счетчики

Далее у нас на очереди числа. Спрашивать о числе во многих отношениях сложнее, чем об адресе электронной почты или веб-адресе. Прежде всего числа не так просты сами по себе. Вот выберите какое-нибудь число. −1? Нет, я имел в виду число из интервала от 1 до 10. 7 1/2? Нет, нет, только не дробь, пожалуйста. А как насчет π? Ну, это совсем уж иррациональный выбор.

Я хочу показать, что о «числах вообще» спрашивают не так часто. Чаще запрашивается число определенного вида (например, целое, но не обыкновенная или десятичная дробь; либо что-либо более сложное, скажем, кратное 10) из определенного диапазона. Возможность сделать все это и предоставляет HTML5.

Рассмотрим такой пример:

```
<input type="number"
       min="0"
       max="10"
       step="2"
       value="6">
```

Будем рассматривать атрибуты по порядку (если хотите, можете в браузере экспериментировать с их значениями).

- `type="number"` — означает, что поле ввода — числовое.
- `min="0"` — говорит, чему равно минимально допустимое значение поля.
- `max="10"` — говорит, чему равно максимально допустимое значение поля.
- `step="2"` — в сочетании со значением `min` определяет множество разрешенных чисел внутри диапазона: 0, 2, 4 и т. д. вплоть до значения `max`.
- `value="6"` — значение по умолчанию. Этот атрибут должен быть вам знаком: он традиционно применяется, чтобы выставить значения по умолчанию в полях формы. (Хочу напомнить, что HTML5 основан на предыдущих версиях HTML. Чтобы делать уже знакомые операции, вам незачем переучиваться со старого стандарта на новый.)

Таков HTML-код поля ввода числа. Надо помнить, что все перечисленные атрибуты необязательны. Если вы почему-либо решили ограничить интервал только минимальным значением, а не максимальным, задайте атрибут `min` и не указывайте `max`. По умолчанию шаг счета равен единице, что позволяет опустить атрибут `step` во всех тех случаях, когда другое значение шага не требуется. Если значение по умолчанию не задано, то атрибут `value` может принять вид пустой строки или вообще исчезнуть.

На этом HTML5 не останавливается. По той же самой милой сердцу цене: ноль целых ноль десятых — программист приобретает следующие замечательные методы JavaScript:

- `input.stepUp(n)` — увеличивает значение поля на *n* единиц;
- `input.stepDown(n)` — уменьшает значение поля на *n* единиц;
- `input.valueAsNumber` — возвращает текущее значение поля в виде числа с плавающей запятой (свойство `input.value`, напомню, всегда строкового типа).

Возникли проблемы с отображением в браузере? Надо сказать, что интерфейс числовых встраиваемых компонентов в различных браузерах реализован по-разному. На iPhone, где проблематичен главным образом пользовательский ввод, браузер вновь оптимизирует виртуальную клавиатуру, на этот раз для ввода чисел, как показано на рис. 9.5.

В десктопной версии Opera то же самое поле `type="number"` отображается в виде счетчика с небольшими стрелками, направленными вверх и вниз. Щелкая на них, можно увеличивать и уменьшать числовое значение (рис. 9.6).

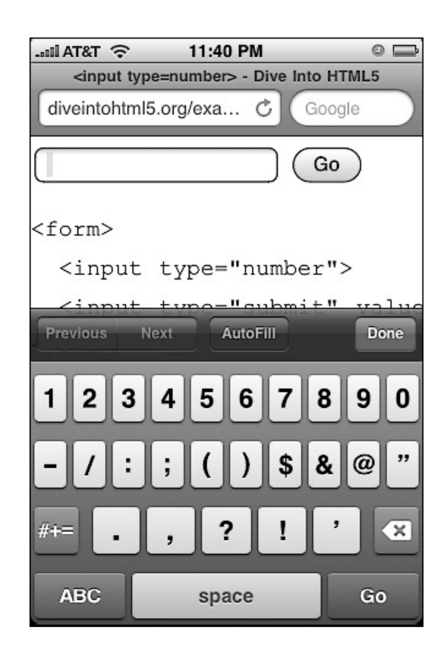


Рис. 9.5. Клавиатура, оптимизированная для ввода числа

Opera бережно относится к атрибутам min, max и step, поэтому вы легко добьетесь приемлемого числового значения. При достижении максимума стрелка «вверх» в счетчике «выцветает» и становится серой (рис. 9.7).

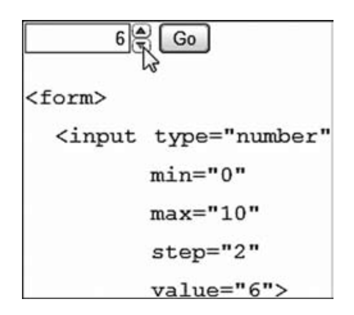


Рис. 9.6. Счетчик

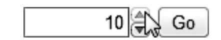


Рис. 9.7. На счетчике достигнуто максимальное значение

Правило, которому подчинены все другие поля ввода, упоминавшиеся в этой главе, распространяется и на type="number": браузер, который не поддерживает этот тип, будет расценивать его как type="text". В поле будет отображено значение по умолчанию (так как оно хранится в атрибуте value), а прочие атрибуты — min, max и др. — будут проигнорированы. Вы можете реализовать их самостоятельно, а можете воспользоваться одним из многочисленных JavaScript-фреймворков, в которых компоненты типа «счетчик» уже реализованы. Но не забудьте сначала протестировать встроенную поддержку HTML5 (см. раздел «Типы полей ввода» главы 2), например, таким образом:

```
if (!Modernizr.inputtypes.number) {
  // встроенной поддержки полей type="number" нет,
  // можно попробовать решить проблему с помощью Dojo
  // или другого JavaScript-фреймворка
}
```

Числа как ползунки

Счетчики, о которых речь шла в предыдущем разделе, — не единственный способ представления вводимых чисел. Вам, наверное, знакомы также компоненты типа «ползунок», внешний вид одного из которых показан на рис. 9.8.

Теперь ползунки доступны и в веб-формах. По своему HTML-коду ползунок фантастически похож на счетчик:

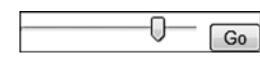


Рис. 9.8. Ползунок

```
<input type="range"
       min="0"
       max="10"
       step="2"
       value="6">
```

Доступны те же самые атрибуты, что и у поля ввода type="number": min, max, step, value. Они и означают то же самое. Отличается только внешний вид пользовательского интерфейса. Как ожидается, вместо обычного поля для ввода браузеры будут отображать type="range" в виде ползунка. Ко времени написания данной главы это уже умели делать последние версии Safari, Chrome и Opera. К сожалению, iPhone выводит на экран просто текстовое поле и даже не оптимизирует экранную клавиатуру для ввода чисел. Все остальные браузеры рассматривают поля такого типа как type="text". В общем, не усматривается серьезных причин, которые мешали бы уже сейчас начать пользоваться ползунками на веб-страницах.

Выборщики даты

В HTML4 не было компонента для выбора даты. Решение было найдено в различных JavaScript-фреймворках (Dojo, jQuery UI, YUI и библиотека Closure), но, конечно, каждое такое решение требует постоянной опоры на фреймворк, с помощью которого построен выборщик.

В HTML5 был в конце концов определен встроенный элемент управления — выборщик даты, не полагающийся на пользовательские сценарии. Фактически есть шесть разных выборщиков: дата, месяц, неделя, время, дата + время и дата + время с часовым поясом.

Из табл. 9.4 видно, что в настоящее время поддержка этой функциональности, мягко скажем, неширока.

Таблица 9.4. Поддержка выборщиков даты

Тип	Opera	Остальные браузеры
type="date"	9.0+	–
type="month"	9.0+	–
type="week"	9.0+	–
type="time"	9.0+	–
type="datetime"	9.0+	–
type="datetime-local"	9.0+	–

Как Opera отображает <input type="date">, показано на рис. 9.9.



Рис. 9.9. Выбор даты

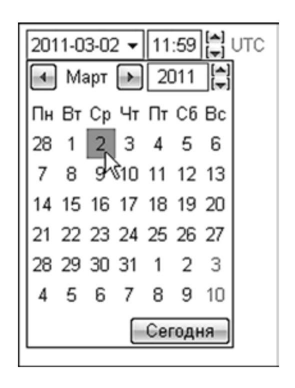


Рис. 9.10. Выбор даты и времени



Рис. 9.11. Выбор месяца

На тот случай, если требуется дата вместе с временем, Opera поддерживает <input type="datetime"> — выборщик, внешний вид которого показан на рис. 9.10.

Если нужны только месяц и год (может иметься в виду, например, месяц, когда истекает срок действия кредитной карты), Opera отобразит объект <input type="month">, показанный на рис. 9.11.

Менее распространен, но также доступен выбор определенной недели года. Это позволяет сделать объект <input type="week"> (рис. 9.12).

Можно, наконец, выбирать только время с помощью объекта <input type="time"> (рис. 9.13).

Вероятно, в конце концов и другие браузеры начнут поддерживать эти типы полей ввода. Поначалу в браузерах, не распознающих type="date" и его варианты, эти поля, как поля type="email" (см. раздел «Адреса электронной почты» этой главы) и других типов, будут отображаться в виде полей ввода обычного текста. Если хотите, <input type="date"> и его «родню» можно уже внедрять: это обрадует пользователей Opera, а другие браузеры тем временем успеют наверстать упущенное. Как вариант, можно, пользуясь <input type="date">, проверять, встроена ли в браузер поддержка выборщиков даты (см. раздел «Типы полей ввода» главы 2), и как запасным решением пользоваться одним из сценариев по вашему выбору:



Рис. 9.12. Выбор недели

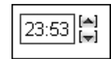


Рис. 9.13. Выбор времени

```
<form>
  <input type="date">
</form>
…
<script>
  var i = document.createElement("input");
  i.setAttribute("type", "date");
  if (i.type == "text") {
    // Встроенной поддержки выборщиков даты нет.
    // Воспользуйтесь Dojo/jQueryUI/YUI/Closure или другим решением, чтобы
    // создать компонент-выборщик и динамически заменить им тег <input>
  }
</script>
```

Формы поиска

Вот действительно интересная тема. Суть всего достаточно проста, объяснять нужно только некоторые детали реализаций. Итак... поиск.

В наши дни это далеко не только поиск от Google или Yahoo! (хотя и это тоже). Вспомните любую форму поиска на любой странице или сайте. Окна поиска имеют Amazon, Newegg, большинство блогов и т. д. Как сделаны эти формы? В их основе, как и в основе большинства полей текстового ввода в современном Интернете, лежит тег <input type="text">. HTML5 исправляет дело:

```
<form>
  <input name="q" type="search">
  <input type="submit" value="Find">
</form>
```

В некоторых браузерах такое поле будет неотличимо от обычного текстового. Но при пользовании Safari на Mac OS X форма поиска будет выглядеть так, как показано на рис. 9.14.

Улавливаете разницу? У поля ввода скругленные углы! Да, конечно, вам сейчас трудно сдержать эмоции, но погодите: это еще не все! Когда вы начнете вводить текст в поле `type="search"`, Safari справа от поля отобразит кнопочку с крестиком. Ее нажатие позволит очистить поле от текста (Google Chrome, по своей технологической начинке родственный Safari, ведет себя так же). Обе эти хитрости созданы по образу «родных» форм поиска в iTunes и других клиентских приложениях Mac OS X (рис. 9.15).

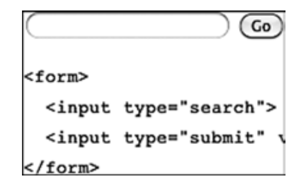




Рис. 9.14. Форма поиска

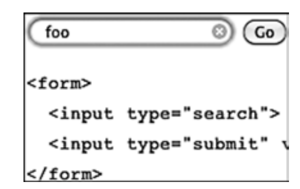


Рис. 9.15. Форма поиска, несущая фокус

Apple применяет тег `<input type="search">` в форме поиска по своему сайту, чтобы на нем чувствовалась атмосфера Macintosh. Но ничего специфического для Macintosh в этой функциональности нет. В сущности, это просто гипертекстовая разметка. Любой браузер на любой платформе может прорисовывать форму поиска в соответствии с общей стилистикой платформы.

Браузеры, которые не воспринимают `type="search"`, будут видеть в нем, как и в остальных новых типах, не более чем обычный `type="text"`. Стоит уже сейчас задуматься об использовании полей `type="search"` во всех окнах поиска.

Выборщики цвета

В спецификации HTML5 описано также поле ввода `<input type="color">`, которое позволяет выбрать цвет и возвращает его шестнадцатеричный код. Браузеры пока не поддерживают выбор цвета — и это, честно говоря, очень плохо. Мне всегда нравились панели выбора цвета в Mac OS. Может быть, когда-нибудь что-то подобное будет реализовано и в браузерах.

СЕКРЕТЫ РАЗМЕТКИ

Есть, впрочем, одна причина, осложняющая применение <input type="search">. К полям поиска Safari не применяет стандартные CSS-стили (стандартными я называю самые базовые вещи: границы, фоновые цвета, фоновые изображения, отступы и пр.). Зато есть круглые уголки!..

И еще об одной вещи

В этой главе я рассказывал о новых типах полей ввода и таких новых функциях, как автофокусировка поля формы. За кадром осталось самое, пожалуй, захватывающее, что можно сообщить об HTML5-формах: автоматическая проверка (валидация) пользовательского ввода. Рассмотрим популярную проблему ввода адреса

электронной почты в веб-форму. Чтобы не случалось сбоев, вы, вероятно, проверяете корректность ввода сначала на клиентской стороне с помощью JavaScript, а потом на серверной стороне с помощью PHP, Python или другого языка серверных сценариев. Проверка электронных адресов с помощью JavaScript чревата двумя большими проблемами:

- неожиданно много посетителей, которые держат JavaScript выключенным (их, вероятно, около 10 %);
- результат проверки будет ошибочным.

Да, в самом деле ошибочным. Определить, является ли валидным адресом электронной почты какая-либо случайная последовательность символов, невероятно трудно (http://www.regular-expressions.info/email.html). Чем напряженнее поиск решения, тем сложнее становится задача (http://www.ex-parrot.com/pdw/Mail-RFC822-Address.html). Я ведь уже сказал, что она очень-очень сложна (http://haacked.com/archive/2007/08/21/i-knew-how-to-validate-an-email-address-until-i.aspx)? Так не проще ли заставить браузер делать всю эту черную работу?

Скриншот на рис. 9.16 взят из Opera 10, хотя нужная функциональность присутствовала уже в Opera 9. Задействован простейший код: атрибуту `type` присвоено значение `email` (см. раздел «Адреса электронной почты» этой главы). Когда пользователь пытается отправить форму с объектом `<input type="email">`, Opera даже при выключенных сценариях автоматически проверяет адрес почты на соответствие RFC.

В Opera предлагается также валидация веб-адресов, вводимых в поля `<input type="url">`, и чисел в полях `<input type="number">`. Валидация чисел принимает во внимание даже атрибуты `min` и `max`, так что Opera не позволит пользователю отослать форму, если введенное в одно из полей число чересчур велико (рис. 9.17).

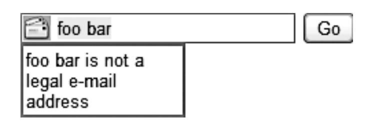


Рис. 9.16. Opera проверяет type="email"

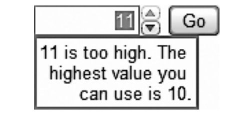


Рис. 9.17. Opera проверяет type="number"

К сожалению, автоматическую валидацию HTML5-форм пока не поддерживает больше ни один браузер, а значит, вам придется еще какое-то время обходиться сценарными альтернативами.

Для дальнейшего изучения

Спецификации и стандарты:

- типы тегов `<input>` (http://bit.ly/akweH4);
- атрибут `<input placeholder>` (http://bit.ly/caGl8N);
- атрибут `<input autofocus>` (http://bit.ly/db1Fj4).

JavaScript-библиотеки: Modernizr (http://www.modernizr.com) — библиотека для тестирования HTML5-функций.

10 Микроданные и другие красивые слова

Приступим

В HTML5 свыше 100 тегов (http://simon.html5.org/html5-elements). Среди них есть как семантические элементы (см. главу 3), так и контейнеры для сценарных API (см. главу 4). На протяжении всей истории HTML (о которой можно прочесть в главе 1) разработчики стандартов спорили, какие теги достойны быть включенными в язык, а какие — нет. Нужен ли в HTML тег `<figure>`? А тег `<person>`? А как насчет тега `<rant>` (по-русски приблизительно «пафосный бред»)? По таким вопросам раз за разом принимаются решения, затем пишутся спецификации, создаются реализации и Интернет спешит навстречу своему будущему.

Конечно, современный HTML не всем по душе, как и любой стандарт. Часть идей в нем не прижилась. Так, в HTML5 отсутствует тег `<person>` и тем более `<rant>`. Ничто не мешает вам включить `<person>` в код вашей веб-страницы, но такая страница не будет валидной. Следует ожидать, что она будет неправильно отображаться в браузерах (см. раздел «Большое отступление о том, как браузеры обрабатывают незнакомые элементы» главы 3) и, возможно, будет противоречить очередной спецификации HTML, если там наконец появится тег `<person>`.

Что же делать веб-мастеру со склонностью к семантической верстке, если вводить собственные теги нельзя? Известны попытки расширить предшествующие версии HTML. Самый популярный способ такого расширения — микроформаты (http://microformats.org), которые пользуются атрибутами `class` и `rel`, определенными в стандарте HTML4. Альтернативный вариант — RDFa (http://www.w3.org/TR/rdfa-syntax/), разработанный для использования с XHTML (см. раздел «Послесловие» главы 1), но в настоящее время переносимый и в HTML. Свои сильные и слабые стороны есть как у микроформатов, так и у RDFa. Это два диаметрально противоположных подхода к решению одной и той же задачи: оснастить веб-страницы дополнительной семантикой, которую нельзя выразить средствами собственно языка HTML. Этой главой я не хотел бы спровоцировать войну между сторонниками двух форматов (иначе было бы просто не обойтись без тега `<rant>`!). Вот почему я сосредоточусь на третьем механизме, входящем в состав HTML5 и тесно интегрированном с ним. Это микроданные.

Что такое микроданные?

Будьте внимательны: в следующем предложении важно каждое слово.

СЕКРЕТЫ РАЗМЕТКИ

Микроданные обогащают DOM парами «имя — значение», взятыми из пользовательских словарей и имеющими определенную область видимости.

Что же это значит? Будем разбираться в обратном порядке, от конца к началу.

Центральное понятие в системе микроданных — *пользовательский словарь*. Множество всех элементов HTML5 — это как бы один словарь, который содержит элементы для представления раздела, статьи (см. раздел «Новые семантические элементы HTML5» главы 3), но не содержит таких элементов, которыми можно было бы представить человека или событие. Если вы желаете представить на веб-странице человека, вам надо определить свой собственный словарь. Это и позволяют сделать микроданные. Каждый может, создав словарь микроданных, начать пользоваться на веб-страницах нестандартными атрибутами.

Далее, о микроданных следует знать, что они оперируют *парами «имя — значение»*. Каждый словарь микроданных объявляет набор именованных свойств. Например, словарь `Person`, описывающий человека, может объявлять свойства `name`, `photo` и т. п. Чтобы на ваших веб-страницах работало то или иное свойство, нужно указать его имя в специально предназначенном для этого месте. В зависимости от того, где конкретно вы объявите имя свойства, механизм микроданных будет по-разному обрабатывать его значение (подробнее об этом читайте в следующем разделе).

В основе системы микроданных лежат не только именованные свойства, но и такое понятие, как *«область видимости»* (scope). Его легко осмыслить, вспомнив, каковы естественные отношения между родительскими и дочерними элементами в DOM. Контейнер `<html>` (см. раздел «Корневой элемент» главы 3) обычно содержит два дочерних тега: `<head>` (см. раздел «Элемент HEAD» главы 3) и `<body>`. У тега `<body>`, как правило, много потомков, и у каждого из них могут быть свои собственные потомки. Так, например, на странице может иметься тег `<h1>` внутри `<hgroup>`, обернутой в `<header>` (см. раздел «Верхние колонтитулы» главы 3) — прямой потомок `<body>`. Точно так же обыкновенная таблица — это теги `<td>` внутри `<tr>` внутри `<table>` (внутри `<body>`). С точки зрения системы микроданных иерархия DOM позволяет заявить: «Все свойства *такого-то* элемента взяты из *такого-то* словаря». Благодаря этому на одной и той же странице можно пользоваться несколькими словарями микроданных. Разрешается даже вкладывать словарь внутрь словаря: все это можно делать благодаря использованию естественной структуры DOM (в этой главе будут показаны примеры вложенных словарей).

Раз уж я заговорил о DOM, то выскажусь по этой теме более развернуто. Микроданные — это способ присвоить дополнительную семантику *тем данным, которые уже присутствуют на веб-странице*. Микроданные разрабатывались не как самостоятельный формат, а как дополнение к HTML. В следующем разделе вы увидите, что применять микроданные лучше всего в том случае, когда возможности HTML используются правильно, но словарю HTML просто не хватает выразитель-

ности. Итак, микроданные — хороший инструмент для «тонкой семантической настройки» тех сведений, которые уже есть в DOM. Если же вы хотите сделать семантическими данные, которых в DOM пока нет, стоит вернуться на шаг назад и задуматься о том, уместно ли использование микроданных.

Стала ли теперь для вас яснее формулировка «секрета», с которого мы начали этот раздел? Надеюсь, да. Теперь посмотрим на микроданные в деле.

Структура микроданных

Определить пользовательский словарь микроданных несложно. Прежде всего для этого потребуется пространство имен, то есть просто URL-адрес. Не требуется строго, чтобы URL пространства имен указывал на работающую веб-страницу. Пусть, например, я решил создать словарь микроданных, который бы описывал человека. Пусть, кроме того, я владею доменом data-vocabulary.org. Тогда в качестве пространства имен для своего словаря я воспользуюсь URL-адресом http://data-vocabulary.org/Person. Итак, достаточно выбрать URL на контролируемом вами домене, чтобы создать вполне уникальный идентификатор.

В словаре мы будем объявлять разные именованные свойства. Начнем с трех базовых свойств:

- name — полное имя пользователя;
- photo — ссылка на картинку с изображением пользователя;
- url — ссылка на сайт, с которым пользователь тем или иным образом связан, например блог или Google-профиль.

Значение первого из этих свойств — обычный текст, а остальных двух — URL-адреса. Не нужно задумываться о микроданных и словарях, чтобы понять, что все это хорошо применимо к страницам, сверстанным естественным образом.

Вообразим, что у вас есть страница-профиль или страница «Обо мне». На ней ваше имя, вероятнее всего, помещено в позицию заголовка, то есть внутрь тега <h1> или подобного. Ваша фотография, чтобы ее могли видеть посетители страницы, заключена в контейнер <img>, а все связанные с вашим профилем URL-адреса имеют вид гиперссылок, на которых посетители могут щелкать. Предположим, что ваш профиль обернут в тег <section> и тем самым отделен от остального содержимого страницы. Пример[1]:

```
<section>
  <h1>Марк Пилгрим</h1>
  <p><img src="http://www.example.com/photo.jpg" alt="[это я улыбаюсь]"></p>
  <p><a href="http://diveintomark.org/">Мой блог</a></p>
</section>
```

Структуру (модель) микроданных образуют пары «имя — значение». Имя свойства микроданных (в нашем примере name, photo или url) всегда объявляется

[1] В этой главе в англоязычных примерах автора последовательно переводились имена и фрагменты связных текстов, а адреса мы оставили в неприкосновенности. — *Примеч. перев.*

в HTML-элементе. Соответствующее значение свойства извлекается из DOM-представления данного элемента. Для большинства элементов HTML значение — всего лишь текстовое содержимое. Есть, однако, несколько нетривиальных случаев, которые перечислены в табл. 10.1.

Таблица 10.1. Откуда извлекаются значения свойств микроданных

Тег	**Значение**
<meta>	Атрибут content
<audio> <embed> <iframe> <img> <source> <video>	Атрибут src
<a> <area> <link>	Атрибут href
<object>	Атрибут data
<time>	Атрибут datetime
Все остальные	Текстовое содержимое

Добавить микроданные на страницу — значит просто добавить несколько новых атрибутов в уже имеющиеся HTML-элементы. Первым делом всегда объявляют, какой именно словарь микроданных используется. Это делает атрибут `itemtype`. Второе, что следует сделать, — объявить область видимости словаря с помощью атрибута `itemscope`. В данном примере все сведения, которые мы хотим семантически разметить, находятся внутри тега `<section>`, а значит, именно в `<section>` мы объявим оба атрибута: `itemtype` и `itemscope`:

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```

Ваше имя — первый фрагмент данных, значимых для нас в контейнере `<section>`. Имя обернуто заголовком `<h1>`, который не требует никакой специальной обработки и попадает в категорию «всех прочих элементов» табл. 10.1. Поэтому значением свойства микроданных окажется просто-напросто текстовое содержимое (если бы имя было обернуто элементом `<p>`, `<div>` или `<span>`, ничего бы не поменялось):

```
<h1 itemprop="name">Марк Пилгрим</h1>
```

По-русски эта запись означает: «Вот свойство `name` словаря микроданных **http://data-vocabulary.org/Person**. Значение свойства — текстовая строка `Марк Пилгрим`».

Следующее на очереди — свойство `photo`. Предполагается, что это URL-адрес. Согласно табл. 10.1, «значением» тега `<img>` является значение в его атрибуте `src`. Но ведь там, в `<img src>`, уже и находится URL фотографии из вашего профиля! Значит, надо просто объявить, что значение свойству `photo` присваивается из тега `<img>`:

```
<p><img itemprop="photo"
        src="http://www.example.com/photo.jpg"
        alt="[это я улыбаюсь]"></p>
```

По-русски это значит: «Вот свойство photo словаря микроданных http://data-vocabulary.org/Person. Значение свойства равно http://www.example.com/photo.jpg».

Наконец, свойство url тоже представляет собой URL-адрес. Согласно табл. 10.1, «значением» тега <a> является значение в его атрибуте href. Это опять идеально соответствует уже существующей разметке. Осталось только сказать, что ваш тег <a> — носитель значения свойства url:

```
<a itemprop="url" href="http://diveintomark.org/">Мой блог</a>
```

По-русски это значит: «Вот свойство url словаря микроданных http://data-vocabulary.org/Person. Значение свойства равно http://diveintomark.org/».

Если сверстанная вами страница выглядит немного иначе, это, конечно, не проблема. Свойства и значения микроданных можно вписывать в любой HTML-код, даже код родом из замшелого XX века, из времен табличной верстки, код в стиле «Да будет проклят тот день, когда я согласился работать с этим кошмаром».

Вообще-то верстать так, как показано ниже, в наши дни отнюдь не рекомендуется. Но подобные конструкции все еще очень популярны, и ничто не мешает «навесить» на них микроданные:

```
<TABLE>
  <TR><TD>Имя<TD>Марк Пилгрим
  <TR><TD>Ссылка<TD>
    <A href=# onclick=goExternalLink()>http://diveintomark.org/</A>
</TABLE>
```

Для присвоения значения свойству name достаточно добавить атрибут itemprop к той клетке таблицы, которая содержит искомое имя. Никаких специальных правил работы с микроданными для ячеек таблицы не определено, поэтому свойству микроданных будет присвоено значение по умолчанию, то есть текстовое содержимое элемента:

```
<TR><TD>Имя<TD itemprop="name">Марк Пилгрим
```

Сложнее будет добавить свойство url. В показанном фрагменте HTML-кода тег <a> используется неправильно. Его атрибут href не содержит ни ссылки на внешний ресурс, ни вообще чего-либо полезного, а для извлечения целевого URL-адреса и перехода по ссылке используется JavaScript-функция в атрибуте onclick (сама функция в коде не показана). Чтобы еще запутать дело, вообразим, что функция goExternalLink(), проходя по ссылке, открывает документ в маленьком всплывающем окне без полосы прокрутки (в прошлом веке в Интернете, право же, было много затейников!).

Даже при этих условиях код можно разметить микроданными, надо лишь проявить немного изобретательности. О прямом использовании тега <a> речи вообще не идет. Адрес, на который указывает ссылка, содержится не в атрибуте href, а правило «В тегах <a> значения свойств микроданных равны значениям атрибутов href» переформулировать нельзя. Но можно воспользоваться элементом-оберткой для всего нашего безобразия и установить значение свойства микроданных url с его помощью:

```
<TABLE itemscope itemtype="http://data-vocabulary.org/Person">
  <TR><TD>Имя<TD>Марк Пилгрим
  <TR><TD>Ссылка<TD>
    <span itemprop="url">
      <A href=# onclick=goExternalLink()>http://diveintomark.org/</A>
    </span>
</TABLE>
```

Поскольку тег <span> не обрабатывается специальным образом, он подчиняется общему правилу: «Значение свойства микроданных равно текстовому содержимому». «Текстовое содержимое» здесь не равносильно всему коду внутри элемента (как было бы, скажем, в DOM-свойстве innerHTML). Речь идет о тексте и только о тексте. В данном случае http://diveintomark.org/ — текстовое содержимое тега <a>, заключенное в контейнер <span>.

Подведем итоги. Свойствами микроданных можно разметить любой код. Если HTML используется правильно, то микроданные добавлять легче, если неправильно, то труднее, но вообще это может быть сделано всегда.

Разметка данных о человеке

Примеры, которые открывают предыдущий раздел, не выдуманы. Для разметки сведений о людях действительно существует словарь микроданных, и он в самом деле настолько просто устроен. Присмотримся к нему ближе.

Чтобы оснастить личный сайт микроданными, проще всего встроить их в страницу «Об авторе». На вашем сайте ведь есть страница «Об авторе», правда? Если нет, следите за тем, как я буду добавлять к образцу такой страницы (http://diveintohtml5.org/examples/person.html) дополнительную семантику. Вот полученный мною результат: http://diveintohtml5.org/examples/person-plus-microdata.html.

Сначала посмотрим на HTML-код, каким он был до внесения в него каких-либо свойств микроданных:

```
<section>
  <img width="204" height="250"
       src="http://diveintohtml5.org/examples/2000_05_mark.jpg"
       alt="[Марк Пилгрим, ок. 2000 г.]">
  <h1>Контактные сведения</h1>
  <dl>
    <dt>Имя</dt>
    <dd>Марк Пилгрим</dd>
    <dt>Должность</dt>
    <dd>Технический евангелист, Google, Inc.</dd>
    <dt>Почтовый адрес</dt>
    <dd>
      100 Main Street<br>
      Anytown, PA 19999<br>
      США
    </dd>
  </dl>
  <h1>Я в сети</h1>
```

```
  <ul>
    <li><a href="http://diveintomark.org/">мой блог</a></li>
     <li><a href="http://www.google.com/profiles/pilgrim">мой профиль Google</a></
li>
     <li><a href="http://www.reddit.com/user/MarkPilgrim">мой профиль Reddit.com</
a></li>
    <li><a href="http://www.twitter.com/diveintomark">мой Twitter</a></li>
  </ul>
</section>
```

Что надо сделать прежде всего, так это объявить, какой словарь используется и какова область видимости каждого из добавляемых свойств. Это делается с помощью атрибутов `itemtype` и `itemscope`, определенных на самом внешнем элементе, то есть содержащем дочерние элементы с информацией, важной для нас. В данном случае таков тег `<section>`:

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```



ПРИМЕЧАНИЕ

За изменениями, которые в этом разделе мы вносим в код страницы, вы можете проследить в режиме онлайн. Было: http://diveintohtml5.org/examples/person.html; стало: http://diveintohtml5.org/examples/person-plus-microdata.html.

Теперь можно начать задавать свойства микроданных из словаря http://data-vocabulary.org/Person. И какие же это свойства? Их список вы увидите, обратившись к самой странице http://data-vocabulary.org/Person. Спецификация микроданных этого не требует, но таков, я бы сказал, «передовой опыт». И если вы желаете, чтобы вашим словарем микроданных стали пользоваться веб-разработчики, то придется его документировать. Где же уместнее расположить документацию, как не по URL-адресу словаря?

В табл. 10.2 перечислены свойства словаря `Person`.

Таблица 10.2. Словарь Person

Свойство	Описание
name	Имя
nickname	Прозвище
photo	Ссылка на изображение
title	Должность человека (например, «Менеджер по финансам»)
role	Профессиональные обязанности человека (например, «Бухгалтер»)
url	Ссылка на веб-страницу (например, на личную страничку человека)
affiliation	Название организации, с которой человек связан (например, фирмы-нанимателя)
friend	Свидетельствует о характере общественного отношения между данным человеком и кем-либо еще
contact	Свидетельствует о характере общественного отношения между этим человеком и кем-либо еще
acquaintance	Свидетельствует о характере общественного отношения между данным человеком и кем-либо еще
address	Местонахождение человека (доступны дочерние свойства street-address, locality, region, postal-code и country-name)

Наш образец страницы «Об авторе» начинается с моей фотографии. Естественно, на фотографию ссылается тег <img>. Объявить, что этот <img> — картинка из моего пользовательского профиля, очень легко; надо лишь добавить атрибут itemprop="photo":

```
<img itemprop="photo" width="204" height="250"
   src="http://diveintohtml5.org/examples/2000_05_mark.jpg"
   alt="[ Марк Пилгрим, ок. 2000 г.]">
```

Что из этого будет присвоено как значение свойству микроданных photo? Значение атрибута src. Из табл. 10.1 можно вспомнить, что «значение» тега <img> приравнивается к значению его атрибута src. У каждого корректно работающего <img> есть атрибут src (иначе фотография будет «битая»), который всегда выражается URL-адресом. Уловили основную мысль? Если пользоваться HTML правильно, то освоить разметку микроданных не составит труда.

Более того, на странице есть не только тег <img>. Это дочерний элемент контейнера <section> — того самого, в который мы только что вписали атрибут itemscope. Отношения между родительскими и дочерними элементами страницы микроданные используют по-своему: для определения области видимости свойств. По-русски мы бы сказали: «Данный тег <section> представляет человека. Все свойства микроданных, определенные в дочерних тегах <section>, — не что иное, как свойства этого человека». Тег <section> можно (если так удобнее) вообразить в роли подлежащего в предложении. Тогда атрибут itemprop будет чем-то вроде сказуемого — «изображен» и т. п., а значение свойства микроданных окажется дополнением:

Этот человек [явное указание <section itemscope itemtype="...">]
изображен [явное указание <img itemprop="photo">]
на фото http://diveintohtml5.org/examples/2000_05_mark.jpg [скрытая ссылка из атрибута <img src>]

«Подлежащее» надо обозначить всего раз, оснастив самый внешний тег <section> атрибутами itemscope и itemtype. «Глагол» задан атрибутом itemprop="photo" в теге <img>. «Дополнение» вообще не следует размечать каким-либо особым образом, потому что из табл. 10.1 известно: свойство микроданных в теге <img> приобретет то же значение, что и атрибут src.

Перейдя к следующему фрагменту кода, мы увидим заголовок <h1> и начало списка определений <dl>. Ни <h1>, ни <dl> размечать микроданными не надо. Вообще далеко не каждому тегу в HTML-коде должно соответствовать свойство микроданных. Ведь микроданные сконцентрированы вокруг свойств, а не заголовков и не технических элементов верстки, которыми свойства окружены. Тег <h1> в нашем примере — не свойство, а всего лишь заголовок. Так же и <dt> со словом Имя — не свойство, лишь метка.

```
<h1>Контактные сведения</h1>
  <dl>
    <dt>Имя</dt>
    <dd>Марк Пилгрим</dd>
```

Сама информация хранится в теге <dd>, на который мы и должны «навесить» атрибут itemprop. Это будет свойство микроданных name; система присвоит ему

значение, равное строке текста внутри <dd>. Нужна ли какая-то особая разметка? Если верить табл. 10.1 — нет: теги <dd> обрабатываются стандартным образом, и значением свойства становится сам текст внутри тега.

Итак, в коде стало:

```
<dd itemprop="name">Марк Пилгрим</dd>
```

По-русски мы бы сказали просто-напросто: «Этого человека зовут Марк Пилгрим». Хорошо, пойдем дальше.

С двумя следующими свойствами дело будет обстоять сложнее. Вот как выглядит HTML-код до нанесения разметки микроданных:

```
<dt>Должность</dt>
<dd>Технический евангелист, Google, Inc.</dd>
```

Взглянув на словарь Person, вы поймете, что текст «Технический евангелист, Google, Inc.» заключает в себе, собственно, два свойства: title («Технический евангелист») и affiliation («Google, Inc.»). Как можно выразить это с помощью микроданных? Вкратце говоря, никак. Микроданные не умеют дробить целую строку текста на самостоятельные свойства. Нельзя сказать: «Первые 18 символов этого текста занесем в одно свойство микроданных, а остальные 12 символов — в другое свойство микроданных».

Однако не все потеряно. Что, если бы вы, например, захотели отображать текст «Технический евангелист» иным шрифтом, чем текст «Google, Inc.»? CSS не предоставляет такого способа. Остается обернуть разные части надписи в «холостые» теги, такие как <span>, и применить к каждому из них различные CSS-правила.

Полезно так же поступать с микроданными. Когда в единой строке содержатся два относительно независимых фрагмента информации, например, как в нашем образце, потенциальные title и affiliation, то достаточно обернуть каждый фрагмент в <span> и каждый <span> объявить особым свойством микроданных:

```
<dt>Position</dt>
<dd><span itemprop="title">Технический евангелист</span> for
   <span itemprop="affiliation">Google, Inc.<span></dd>
```

Ура! Вот то же самое в переводе на русский: «Должность этого человека — технический евангелист. Наниматель этого человека — компания Google, Inc.». Два предложения — два свойства микроданных. Разметка чуть усложнилась, но это усложнение оправданно.

Схожее решение применяется и для разметки данных об адресах. В словаре Person объявлено свойство address, которое выступает и как самостоятельная единица микроданных. У свойства address собственный словарь (**http://data-vocabulary.org/Address**). Ему подчинено несколько дочерних свойств: street-address, locality, region, postal-code и country-name.

Если вы программист, то вам, вероятно, знакома «точечная» нотация, в которой принято описывать объекты и их свойства. Рассмотрим следующий набор:

- Person;
- Person.address;
- Person.address.street-address;

- Person.address.locality;
- Person.address.region;
- Person.address.postal-code;
- Person.address.country-name.

В нашем примере весь адрес содержится в единственном теге <dd> (напомню, тег <dt> — просто метка, в расширении семантики с помощью микроданных он никакой роли не играет). Значит, можно было бы записать свойство address без ухищрений, просто добавив атрибут itemprop в тег <dd>:

```
<dt>Почтовый адрес</dt>
<dd itemprop="address">
```

Но не стоит забывать о том, что свойство address — самостоятельная единица микроданных. Теперь нам снова пригодятся атрибуты itemscope и itemtype:

```
<dt>Почтовый адрес</dt>
<dd itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
```

Все это мы уже видели, но только применительно к единицам самого верхнего уровня. Атрибуты itemtype и itemscope были определены в теге <section>, и все дочерние теги <section>, оснащенные свойствами микроданных, находились в «области видимости» словаря Person. Сейчас мы впервые сталкиваемся с вложенными областями видимости: новые itemtype и itemscope (в теге <dd>) переопределяют существующие атрибуты (в теге <section>). Система вложенных областей видимости работает точно так же, как HTML DOM. У тега <dd> есть сколько-то дочерних элементов, и все они входят в область видимости словаря, объявленного в <dd>. Но как только тег <dd> закрывается соответствующей конструкцией </dd>, мы возвращаемся в область видимости словаря, который определен в родительском теге (в данном случае <section>).

Словарь микроданных Address страдает той проблемой, которую мы уже рассмотрели на примере свойств title и affiliation. Адрес — одна большая строка, которую мы хотим разбить на несколько самостоятельных свойств микроданных. Решение прежнее — обернуть каждый информативный фрагмент в <span> и поставить каждому <span> в соответствие по одному свойству микроданных:

```
  <dd itemprop="address" itemscope
      itemtype="http://data-vocabulary.org/Address">
    <span itemprop="street-address">100 Main Street</span><br>
    <span itemprop="locality">Anytown</span>,
    <span itemprop="region">PA</span>
    <span itemprop="postal-code">19999</span>
    <span itemprop="country-name">USA</span>
  </dd>
</dl>
```

Итак, по-русски: «У этого человека есть почтовый адрес. Та часть адреса, которая содержит номер дома и название улицы, выглядит как 100 Main Street. Населенный пункт называется Anytown. Код региона — PA (штат Пенсильвания). Почтовый индекс — 19999, а страна — США». Все проще простого.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Специфичен ли для США описанный выше формат почтовых адресов?

Ответ: Нет. Свойства словаря Address в достаточной мере обобщенные и способны описывать большинство почтовых адресов в мире. Не во всех адресах значение будет присваиваться в точности каждому свойству, но это не страшно. Есть адреса, которые, возможно, потребуют в каких-либо свойствах более чем по одной «строке»; это тоже не страшно. Так, например, если в состав вашего почтового адреса входят название улицы, номер дома и номер квартиры, все это без различия попадет в свойство street-address:

```
<p itemprop="address" itemscope
    itemtype="http://data-vocabulary.
org/Address">
  <span itemprop="street-address">
    100 Main Street
    Suite 415
  </span>
  ...
</p>
```

На странице «Об авторе», выступающей в этом примере как образец, есть еще список URL-адресов. В словаре Person задано соответствующее свойство под именем url. Это, в сущности, может быть любой веб-адрес (хотя и обязательно веб-адрес, как вы понимаете). Я хочу сказать, что определение свойства url чрезвычайно расплывчато. Под него подходит любой URL-адрес, который вам угодно связывать с человеком: адрес блога, фотогалереи, профиля на таком стороннем сайте, как Facebook и Twitter.

Важно заметить здесь, что одному человеку может быть поставлено в соответствие несколько свойств url. Говоря технически, значение каждого свойства может быть определено более чем однажды. Но вплоть до настоящего момента мы не знали, как применить эту возможность с пользой для себя. Можно, оказывается, держать две фотографии (в свойствах photo) с разными URL-источниками картинок. В данном примере четыре разных URL-адреса: мой блог, мой профиль в Google, мой пользовательский профиль Reddit.com и моя учетная запись Twitter — будут перечислены единым списком. С точки зрения HTML-кода имеем список из четырех ссылок: четыре тега <a>, на каждый из которых приходится собственный пункт списка <li>. С точки зрения разметки микроданных каждому тегу <a> присвоен атрибут itemprop="url":

```
<h1>Я в сети</h1>
<ul>
  <li><a href="http://diveintomark.org/"
         itemprop="url">мой блог</a></li>
  <li><a href="http://www.google.com/profiles/pilgrim"
         itemprop="url">мой профиль Google</a></li>
  <li><a href="http://www.reddit.com/user/MarkPilgrim"
         itemprop="url">мой профиль Reddit.com</a></li>
  <li><a href="http://www.twitter.com/diveintomark"
         itemprop="url">мой Twitter</a></li>
</ul>
```

Согласно табл. 10.1, теги <a> обрабатываются особым образом: значение свойства микроданных приравнивается к значению атрибута href, а не к вложенному

текстовому содержимому. Таким образом, вовсе не обращая внимания на текст каждой ссылки, процессор микроданных узнает (в переводе на русский язык) следующее: «У этого человека есть URL-адрес http://diveintomark.org/. Еще у этого человека есть URL-адрес http://www.google.com/profiles/pilgrim, а также URL-адрес http://www.reddit.com/user/MarkPilgrim и URL-адрес http://www.twitter.com/diveintomark».

Знакомство с Google Rich Snippets. Вернемся на шаг назад и спросим себя, зачем все это нужно. Не добавляем ли мы новую семантику просто ради того, чтобы добавить новую семантику? Конечно, нет. Не следует думать, будто мне больше вашего нравится возиться с угловыми скобочками. Но почему именно микроданные и зачем вся эта суматоха вокруг них? Есть два больших класса приложений, способных работать с микроданными HTML5:

- браузеры;
- поисковые системы.

Для браузеров в спецификации HTML5 определен набор функций DOM API, которые позволяют извлекать из веб-страниц целостные объекты микроданных, свойства и значения свойств. На момент написания этих строк ни один браузер не поддерживал данный API. Вообще ни один. Это, пожалуй, тупик, по крайней мере до тех пор, пока производители браузеров не задумаются о реализации соответствующего API на клиентской стороне.

Другой крупный «потребитель» HTML-кода — поисковики. Что может поисковая система сделать со свойствами микроданных, описывающими человека? А если отображать в выдаче не просто название страницы и фрагмент ее текста, а какую-либо комбинацию наших структурированных данных? Например, полное имя, название должности, наниматель, адрес, возможно, уменьшенная, в режиме предварительного просмотра, пользовательская картинка. Привлекло бы это ваше внимание? Мое — да.

Google поддерживает микроданные в рамках программы Rich Snippets (http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170). Когда программа-краулер Google разбирает страницу и находит при этом свойства микроданных, имена которых удовлетворяют словарю http://data-vocabulary.org/Person, значения таких свойств извлекаются и сохраняются особо. Google предоставляет удобный инструмент для визуализации того, как поисковая система «видит» свойства микроданных. Наша страница «Об авторе», оснащенная разметкой микроданных (http://diveintohtml5.org/examples/person-plus-microdata.html), с точки зрения поисковой системы выглядит так:

```
Item
  Type: http://data-vocabulary.org/person
  photo = http://diveintohtml5.org/examples/2000_05_mark.jpg
  name = Марк Пилгрим
  title = Технический евангелист
  affiliation = Google, Inc.
  address = Item( 1 )
  url = http://diveintomark.org/
  url = http://www.google.com/profiles/pilgrim
```

```
  url = http://www.reddit.com/user/MarkPilgrim
  url = http://www.twitter.com/diveintomark
Item 1
  Type: http://data-vocabulary.org/address
  street-address = 100 Main Street
  locality = Anytown
  region = PA
  postal-code = 19999
  country-name = USA
```

Здесь все: свойство `photo` из атрибута `<img src>`, все четыре URL-адреса из атрибутов `<a href>` перечисленных списком ссылок и даже адрес (выступающий здесь как `Item 1`) со всеми пятью его дочерними свойствами.

Как же Google пользуется всей этой информацией? Бывают разные случаи. Вопрос о том, как отображать свойства микроданных, какие именно из них и следует ли их вообще отображать, лишен жесткой регламентации. Если Google определит, что наша страница «Об авторе» должна войти в результаты выдачи при поиске на `Mark Pilgrim`, и система сочтет, что оригинальную разметку микроданных на нашей странице можно показать пользователю, то поисковая выдача может выглядеть приблизительно так, как показано на рис. 10.1.

About Mark Pilgrim
Anytown PA - Developer advocate - Google, Inc.
Excerpt from the page will show up here.
Excerpt from the page will show up here.
diveintohtml5.org/examples/person-plus-microdata.html - Cached - Similar pages

Рис. 10.1. Образец поисковой выдачи, содержащей страницу, на которой информация о человеке размечена микроданными

Первая строка — это заглавие страницы, содержащееся в теге `<title>`. Ничего удивительного — так Google поступает со всеми страницами. Вторая строка полна сведений из разметки микроданных, которой мы оснастили страницы. `Anytown PA` — часть почтового адреса, размеченного как одно из свойств словаря http://data-vocabulary.org/Address. `Developer advocate` и `Google, Inc.` — значения двух свойств из словаря http://data-vocabulary.org/Person (`title` и `affiliation` соответственно).

Этот опыт свидетельствует о том, что для оптимизации поисковой выдачи не нужно представлять крупную корпорацию и специально, за деньги, договариваться с командой разработчиков поисковой системы. Достаточно потратить десять минут на добавление кое-каких HTML-атрибутов в данные, которые вы уже и так опубликовали.

РАЗМЕТКА В ВОПРОСАХ И ОТВЕТАХ

Вопрос: Я сделал все, как вы сказали, но поисковая выдача Google не изменилась. Что не так?

Ответ: «Google не может гарантировать, что разметка микроданных, присутствующая на какой-либо странице

или сайте, будет использована в результатах поиска» (http://www.google.com/support/webmasters/bin/answer.py?hl= en&answer=99170). Даже если Google не обратит внимания на вашу разметку микроданных, мимо нее может не пройти другая поисковая система. Ведь микроданные, как и все прочее в спецификации HTML5, определяются открытым стандартом, который может реализовать любой желающий. Ваша задача — предоставить максимум данных, а другие пусть, в свою очередь, думают, что с этими данными делать. Возможно, вас приятно удивят их находки!

Разметка данных об организации

Микроданные не привязаны к одному-единственному словарю. «Об авторе», конечно, хорошая страница, но у вас такая страница, скорее всего, одна. Жаждете нового опыта? Научимся оснащать микроданными страницы организаций и предприятий.

Я создал несложный образец страницы данных об учреждении (http://diveintohtml5.org/examples/organization.html). Рассмотрим первоначальный HTML-код этой страницы, без разметки микроданных:

```
<article>
  <h1>Google, Inc.</h1>
  <p>
    1600 Amphitheatre Parkway<br>
    Mountain View, CA 94043<br>
    США
  </p>
  <p>650-253-0000</p>
  <p><a href="http://www.google.com/">Google.com</a></p>
</article>
```



ПРИМЕЧАНИЕ

За изменениями, которые в этом разделе мы вносим в код страницы, вы можете проследить в режиме онлайн. Было: http://diveintohtml5.org/examples/organization.html; стало: http://diveintohtml5.org/examples/organization-plus-microdata.html.

Коротко и хорошо. Все сведения об организации содержатся внутри тега <article>, с которого мы и начнем преобразования:

```
<article itemscope itemtype="http://data-vocabulary.org/Organization">
```

Как и при разметке данных о человеке, надо добавить атрибуты itemscope и itemtype в самый внешний тег, каковым в данном случае является <article>. В атрибуте itemtype надо объявить используемый словарь микроданных (сейчас это http://data-vocabulary.org/Organization). Атрибут itemscope удостоверяет, что все свойства микроданных, определенные в элементах-потомках <article>, относятся к тому же словарю.

Что же представляет собой словарь Organization? Он очень прост и даже прямолинеен. Часть его должна быть уже знакома вам. Внимание на табл. 10.3.

Таблица 10.3. Словарь Organization

Свойство	**Описание**
name	Название организации
url	Ссылка на сайт организации
address	Местонахождение организации (может содержать дочерние свойства street-address, locality, region, postal-code, country-name)
tel	Телефонный номер организации
geo	Географические координаты организации (всегда с двумя дочерними свойствами: latitude и longitude)

Внутри самой внешней обертки <article> первый элемент нашей разметки — <h1>. В заголовке <h1> содержится название организации, поэтому именно сюда следует вписать атрибут itemprop="name":

```
<h1 itemprop="name">Google, Inc.</h1>
```

Согласно табл. 10.1, теги <h1> обрабатываются стандартным образом и значение свойства микроданных, определенного в таком теге, равно текстовому содержимому тега. По-русски мы сказали бы всего лишь: «Название организации — Google, Inc.».

Следующим по порядку идет фрагмент адреса (улица и дом). Адреса организаций размечаются в точности так же, как и адреса людей. Надо сначала добавить атрибут itemprop="address" в самый внешний элемент, дочерними по отношению к которому будут все части адреса (в данном случае таков тег <p>). Этим способом мы введем свойство address словаря Organization.

Надо объявить атрибуты itemtype и itemscope, которые бы свидетельствовали, что у данной единицы, определяемой словарем Address, есть несколько дочерних свойств:

```
<p itemprop="address" itemscope
   itemtype="http://data-vocabulary.org/Address">
```

Наконец, каждый фрагмент сведений, для которого подыскивается соответствующее свойство словаря Address (street-address, locality, region, postal-code, country-name), надо обернуть тегом <span> и после этого вписать нужное свойство:

```
<p itemprop="address" itemscope
   itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">1600 Amphitheatre Parkway</span><br>
  <span itemprop="locality">Mountain View</span>,
  <span itemprop="region">CA</span>
  <span itemprop="postal-code">94043</span><br>
  <span itemprop="country-name">USA</span>
</p>
```

По-русски мы сказали бы: «У организации есть адрес, хранимый как совокупность нескольких свойств. Свойство с названием улицы и номером дома равно 1600 Amphitheatre Parkway. Свойство с названием населенного пункта равно Mountain View. Свойство с названием региона равно CA (это штат Калифорния). Почтовый индекс — 94043, а страна — США».

Теперь о телефонном номере организации. Всем известно, как сложно построены телефонные номера. В каждой стране их синтаксис какой-то свой, и при международных звонках случаются накладки. В данном примере есть телефонный номер США в формате, удобном для звонящих из любой части США:

```
<p itemprop="tel">650-253-0000</p>
```

Если вы вдруг не заметили, область видимости словаря Address закончилась с закрытием соответствующего тега <p>. Теперь мы вновь определяем свойства словаря Organization.

Чтобы перечислить несколько телефонных номеров, например один для клиентов из США, а другой для международных звонков, поступайте следующим образом. Поскольку каждое свойство микроданных может быть повторено, достаточно заключить оба телефона в независимые HTML-элементы, отделив их от текстовых меток:

```
<p>
  Для клиентов из США: <span itemprop="tel">650-253-0000</span><br>
  Для клиентов из Англии: <span itemprop="tel">00 + 1* + 6502530000</span>
</p>
```

Согласно табл. 10.1, теги <p> и <span> обрабатываются стандартным образом. Значение свойства микроданных tel, определенного в таком элементе, — это просто текстовое содержимое. Словарь микроданных Organization не предусматривает деления телефонного номера на части; значение свойства tel — текст в свободной форме. Если хотите указать в скобках код страны или области, а группы цифр разделить не дефисами, а пробелами, это можно сделать. Как будет читать телефонные номера клиентская программа — обработчик микроданных, решать самой программе.

Следующее свойство, url, опять знакомо нам. Как описано в предыдущем разделе, можно связывать URL-адрес с человеком (Person), но можно, естественно, и с организацией. Как URL-ресурс может выступать главная страница сайта фирмы, страница контактов, страница одного из продуктов и т. п. Ссылку на сетевой ресурс компании (или о компании) снабжают атрибутом itemprop="url":

```
<p><a itemprop="url" href="http://www.google.com/">Google.com</a></p>
```

Согласно табл. 10.1, теги <a> обрабатываются особым образом: значение свойства микроданных приравнивается к значению атрибута href, а не к вложенному текстовому содержимому. По-русски: «Эта организация как-то связана с URL-адресом http://www.google.com/». Ничего более конкретного о характере связи сказать нельзя; текст ссылки — Google.com — игнорируется.

Осталось сказать о геолокации, но не о W3C API геолокации (о котором см. главу 6), а о том, как отразить в разметке микроданных физическое местонахождение организации.

Вплоть до настоящего момента все наши примеры вращались вокруг дополнительной разметки данных, *видимых* на экране. Так, например, если есть <h1> с текстом — названием фирмы, мы добавим в тег <h1> атрибут itemprop, чтобы тем самым объявить: видимый на экране текст (заголовок) представляет собой название фирмы. Другой пример: если тег <img> ссылается на фотографию, мы тоже добавим

в этот тег атрибут itemprop, который будет свидетельствовать: видимая на экране картинка представляет собой фотографию человека.

Не таковы геолокационные сведения. Широта и долгота местонахождения офиса организации (с точностью до четырех десятичных знаков!) на экране отсутствуют. Да и наша страница-образец до оснащения разметкой микроданных вовсе не содержала геолокационных сведений. На странице есть ссылка на Карты Google, но в URL-адрес этой ссылки не входят координаты — широта и долгота (вернее, входят, но в формате, специфичном для Google). Если бы даже на нашей странице стояла ссылка на гипотетический веб-сервис с картами, который бы принимал широту и долготу как параметры URL-адреса динамически генерируемой страницы, нельзя было бы выделить нужные фрагменты адреса и разметить их микроданными. Невозможно объявить, что первый параметр URL-запроса представляет широту, второй — долготу, а все остальные параметры несущественны.

Для сложных случаев, таких как этот, в HTML5 предусмотрен механизм аннотирования *невидимых* данных. Пользоваться им следует лишь при отсутствии альтернатив. Если важные для вас данные можно хоть как-то вывести на экран, это обязательно надо сделать. Невидимым машиночитаемым данным свойственно очень быстро «портиться». Это связано с тем, что, обновляя видимый текст на страницах, очень часто забывают обновить машиночитаемые данные, причем чаще, чем вам кажется (если вам пока не приходилось проявлять подобную забывчивость, то, думаю, еще придется).

Так или иначе в некоторых случаях нельзя обойтись без невидимых данных. Пусть, например, ваш начальник хочет, чтобы на сайте были машиночитаемые сведения о координатах офиса, но чтобы при этом пользовательский интерфейс не загромождала пара непонятных шестизначных чисел. В такой ситуации невидимые данные — единственная альтернатива. Можно лишь немного себя обезопасить, если поместить машиночитаемые сведения сразу после видимого текста, который они описывают. Тогда, возможно, при обновлении данных на странице администратор сайта не забудет внести правку не только в текст, отображаемый на экране, но и в данные, следующие за ним в коде страницы. В нашем примере можно создать внутри того же тега <article>, который содержит остальные свойства организации, тег <span> и внести в него невидимые геолокационные сведения:

```
  <span itemprop="geo" itemscope
        itemtype="http://data-vocabulary.org/Geo">
    <meta itemprop="latitude" content="37.4149" />
    <meta itemprop="longitude" content="-122.078" />
  </span>
</article>
```

Для геолокации существует свой собственный словарь микроданных, как, например, для адреса человека или организации. Вот почему в этом теге <span> должны присутствовать следующие три атрибута:

- itemprop="geo" — показывает, что в данном элементе определено свойство организации geo;
- itemtype="http://data-vocabulary.org/Geo" — объявляет словарь микроданных, которому соответствуют свойства внутри самого этого элемента;

- itemscope — удостоверяет, что этот элемент соответствует единице микроданных со словарем, объявленным в атрибуте itemtype. Все свойства, находящиеся внутри, принадлежат словарю микроданных Geo (http://data-vocabulary.org/Geo), а не Organization (http://data-vocabulary.org/Organization).

В нашем примере решена и еще одна проблема, а именно: «Как аннотировать невидимые данные?» Это делается с помощью тега <meta>. В предыдущих версиях HTML разрешалось использовать тег <meta> только в начальном разделе страницы — <head> (см. раздел «Элемент HEAD» главы 3). А вот в HTML5 тег <meta> можно помещать куда угодно. Так мы и поступаем:

```
<meta itemprop="latitude" content="37.4149" />
```

Согласно табл. 10.1, тег <meta> обрабатывается специальным образом: значение свойства микроданных приравнивается к значению атрибута content. Поскольку этот атрибут никогда не отображается внешним образом, мы приобретаем власть над неограниченно большим объемом невидимых данных. Но большая власть — это и большая ответственность. На нашей совести будет синхронизация невидимых данных с видимым текстом вокруг них.

В системе Google Rich Snippets нет непосредственной поддержки словаря Organization, поэтому красивой поисковой выдачи я здесь не покажу. Но с представлением организаций тесно связаны следующие два типа разметки микроданных, которые, в свою очередь, поддерживаются Google Rich Snippets.

Разметка данных о событии

Событиям свойственно происходить. Некоторым событиям свойственно происходить в заранее назначенное время. Разве не хорошо было бы, если бы поисковые системы могли узнавать о предстоящих событиях непосредственно от создателей веб-страниц? А между тем микроданные предоставляют такую возможность.

Рассмотрим фрагмент графика моих выступлений на конференциях (http://diveintohtml5.org/examples/event.html):

```
<article>
  <h1>День разработчика Google (2009)</h1>
  <img width="300" height="200"
       src="http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg"
       alt="[Марк Пилгрим на трибуне]">
  <p>
    День разработчика Google (Google Developer Day) — это шанс узнать
    о продуктах Google для разработчиков прямо из первых рук, от инженеров,
    создавших эти продукты. На однодневной конференции запланированы
    семинары и "офисные часы", посвященные таким веб-технологиям, как Google
    Maps, OpenSocial, Android, AJAX API, Chrome и Google Web Toolkit.
  </p>
  <p>
    <time datetime="2009-11-06T08:30+01:00">6 ноября 2009 г., 8:30</time>
      &ndash;
    <time datetime="2009-11-06T20:30+01:00">20:30</time>
```

```
  </p>
  <p>
    Конгресс-центр<br>
    5th května 65<br>
    140 21 Прага 4<br>
    Чехия
  </p>
  <p>
    <a href="http://code.google.com/intl/cs/events/developerday/2009/home.html">
    Главная страница GDD в Праге</a></p>
</article>
```



ПРИМЕЧАНИЕ

За изменениями, которые в этом разделе мы вносим в код страницы, вы можете проследить в режиме онлайн. Было: http://diveintohtml5.org/examples/event.html; стало: http://diveintohtml5.org/examples/event-plus-microdata.html.

Вся информация о событии содержится в теге <article>, куда мы и поместим атрибуты itemtype и itemscope:

```
<article itemscope itemtype="http://data-vocabulary.org/Event">
```

Пространство имен словаря Event задано URL-адресом http://data-vocabulary.org/Event. По названному адресу доступна симпатичная таблица с описанием свойств словаря. Перечислю эти свойства и здесь (табл. 10.4).

Таблица 10.4. Словарь Event

Свойство	Описание
summary	Название события
url	Ссылка на страницу подробной информации о событии
location	Место, где проходит событие, и место сбора участников. Если необходимо, может быть представлено вложенной единицей словаря Organization (см. раздел «Разметка данных об организации» данной главы) или Address (см. раздел «Разметка данных о человеке» этой главы)
description	Описание события
startDate	Дата и время начала события (в ISO-формате)
endDate	Дата и время окончания события (в ISO-формате)
duration	Продолжительность события (в ISO-формате для продолжительности)
eventType	Категория события (например, «Концерт» или «Лекция»). Предполагается текст в свободной форме, а не атрибут из закрытого списка
geo	Географические координаты места события (всегда с двумя дочерними свойствами: latitude и longitude)
photo	Ссылка на изображение, в частности фотографию, связанную с событием

Название события заключено в нашем примере в тег <h1>. Согласно табл. 10.1, теги <h1> обрабатываются стандартным образом и значение свойства микроданных, определенного в таком элементе, равно текстовому содержимому элемента. Значит, достаточно добавить атрибут itemprop, который будет показывать, что данный тег <h1> содержит название события:

```
<h1 itemprop="summary">День разработчика Google (2009)</h1>
```

По-русски: «Название этого события — День разработчика Google (2009)».

На странице данных о событии есть фотография, которую можно разметить свойством photo. Как и следовало ожидать, фотография заключена в тег <img>. Как и свойство photo словаря Person (см. раздел «Структура микроданных» этой главы), свойство photo словаря Event имеет своим значением URL-адрес. Поскольку из табл. 10.1 известно, что значение свойства микроданных, определенного в теге <img>, приравнивается к значению атрибута src, нам достаточно вписать в <img> атрибут itemprop:

```
<img itemprop="photo" width="300" height="200"
     src="http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg"
     alt="[Марк Пилгрим на трибуне]">
```

По-русски: «С этим событием связана фотография http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg».

Затем следует подробное описание нашего события — абзац текста в свободной форме:

```
<p itemprop="description">
  День разработчика Google (Google Developer Day) — это шанс узнать
  о продуктах Google для разработчиков прямо из первых рук, от инженеров,
  создавших эти продукты. На однодневной конференции запланированы
  семинары и "офисные часы", посвященные таким веб-технологиям, как Google
  Maps, OpenSocial, Android, AJAX API, Chrome и Google Web Toolkit.</p>
```

А теперь нечто новое. Событиям свойственно происходить в определенные числа, начинаться и заканчиваться в определенное время. В HTML5, как известно, для обозначения даты и времени существует тег <time> (см. раздел «Дата и время» главы 3), чем мы и воспользовались. Возникает такой вопрос: как разметить теги <time> микроданными?

Заглянув в табл. 10.1, мы увидим, что тег <time> обрабатывается особым образом. Значение свойства микроданных, определенного в теге <time>, равно значению атрибута datetime. Между тем свойства startDate и endDate словаря Event требуют того же ISO-формата дат, что и свойство datetime тега <time>. В этом примере семантика основного словаря HTML и семантика нашего (пользовательского) словаря микроданных отлично дополняют друг друга. Итак, чтобы обозначить с помощью микроданных начало и конец события, надо, во-первых, правильно использовать возможности HTML (для оформления даты и времени применять тег <time>), а во-вторых, добавить внутрь <time> атрибут itemprop:

```
<p>
   <time itemprop="startDate" datetime="2009-11-06T08:30+01:00">6 ноября 2009 г.,
8:30</time>
    &ndash;
  <time itemprop="endDate" datetime="2009-11-06T20:30+01:00">20:30</time>
</p>
```

По-русски: «Это событие начинается 6 ноября 2009 года в 8:30 утра и длится до 20:30 того же дня (время пражское, GMT+1)».

Теперь о свойстве location. Словарь Event предусматривает, что это свойство может быть выражено самостоятельной единицей словаря Organization или Address.

В нашем случае событие имеет место в пражском Конгресс-центре, который специализируется на проведении конференций различного рода. Если воспользоваться разметкой организации, то мы сможем указать как название, так и адрес.

Сначала объявим, что тег <p> с адресом — это свойство события location и что этот тег также является самостоятельной единицей микроданных, принадлежащей словарю http://data-vocabulary.org/Organization:

```
<p itemprop="location" itemscope
   itemtype="http://data-vocabulary.org/Organization">
```

Затем обернем название организации в тег <span> и «навесим» на него атрибут itemprop:

```
<span itemprop="name">Конгресс-центр</span><br>
```

Свойство "name" находится в словаре Organization, а не в Event. С тега <p> начинается область видимости словаря Organization, а закрывающего тега </p> еще не было. Все объявляемые здесь свойства микроданных принадлежат словарю, область видимости которого была объявлена последней по счету и еще длится. Вложенные словари можно представить себе в виде стека. До тех пор, пока самый верхний элемент стека не удален, следуют свойства, принадлежащие словарю Organization.

В сущности, мы сейчас добавим в стек еще и третий словарь: адрес (Address) организации (Organization), принимающей событие (Event):

```
<span itemprop="address" itemscope
      itemtype="http://data-vocabulary.org/Address">
```

Мы снова хотим занести отдельные части адреса в самостоятельные свойства микроданных, а значит, понадобится несколько тегов <span>, в которых мы определим соответствующие атрибуты itemprop (если вы не успеваете за ходом изложения, то вернитесь к разделам «Разметка данных о человеке» и «Разметка данных об организации» этой главы, где можно прочитать об аннотировании адресов людей и организаций микроданными).

```
<span itemprop="street-address">5th května 65</span><br>
<span itemprop="postal-code">140 21</span>
<span itemprop="locality">Прага 4</span><br>
<span itemprop="country-name">Чехия</span>
```

У адреса других свойств не предусмотрено. Мы закроем тег <span> (область видимости словаря Address), тем самым убирая из стека верхний элемент:

```
</span>
```

У организации других свойств тоже не предусмотрено, что позволяет нам закрыть тег <p> (область видимости словаря Organization) и тем самым убрать из стека еще один элемент:

```
</p>
```

Теперь мы возвращаемся к разметке свойств события (Event). Следующее свойство, geo, представляет физический адрес события. В нем используется словарь Geo, которым мы уже пользовались для оформления физического адреса организации

(в предыдущем разделе). Опять нужен <span> в качестве контейнера, который будет содержать атрибуты itemtype и itemscope. Внутри <span> будут присутствовать два тега <meta>, один с широтой (свойство latitude) и один с долготой (свойство longitude):

```
<span itemprop="geo" itemscope itemtype="http://data-vocabulary.org/Geo">
  <meta itemprop="latitude" content="50.047893" />
  <meta itemprop="longitude" content="14.4491" />
</span>
```

После того как мы закроем <span> со свойствами из словаря Geo, мы возвращаемся к свойствам из словаря Event. Последнее оставшееся свойство уже знакомо нам: это url. Связать URL-адрес с событием можно точно таким же образом, как с человеком (см. раздел «Разметка данных о человеке» этой главы) и с организацией (см. раздел «Разметка данных об организации» этой главы). При корректном использовании HTML, то есть если гиперссылка имеет вид <a href>, достаточно добавить к ссылке атрибут itemprop="url":

```
<p>
  <a itemprop="url"
    href="http://code.google.com/intl/cs/events/developerday/2009/home.html">
    Главная страница GDD в Праге
  </a>
</p>
</article>
```

На странице, взятой нами за образец (http://diveintohtml5.org/examples/event.html), присутствует и второе событие: мое выступление на конференции ConFoo в Монреале. Ради экономии места я не буду подробно рассматривать ее разметку, которая по существу ничем не отличается от разметки данных о конференции в Праге: событие (единица словаря Event) с вложенными сведениями о геолокации (единица словаря Geo) и адресе (Address). Говорю об этом лишь затем, чтобы повторить: на одной странице может быть отражено несколько событий, каждое — в собственной разметке микроданных.

И снова Google Rich Snippets. Тестовый инструмент Google Rich Snippets утверждает, что из нашей страницы события (http://diveintohtml5.org/examples/event-plus-microdata.html) поисковые роботы Google сумели извлечь следующую информацию:

```
Item
  Type: http://data-vocabulary.org/Event
  summary = День разработчика Google (2009)
  eventType = конференция
  photo = http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg
  description = День разработчика Google (Google Developer Day) - это шанс
                узнать о продуктах Google для разработчиков прямо из первых
                рук, от инженеров, создавших эти продукты. На однодневной
                конференции запланированы семинары и "офисные часы",
                посвященные таким веб-технологиям…
  startDate = 2009-11-06T08:30+01:00
  endDate = 2009-11-06T20:30+01:00
```

```
  location = Item(__1)
  geo = Item(__3)
  url = http://code.google.com/intl/cs/events/developerday/2009/home.html
Item
  Id: __1
  Type: http://data-vocabulary.org/Organization
  name = Конгресс-центр
  address = Item(__2)
Item
  Id: __2
  Type: http://data-vocabulary.org/Address
  street-address = 5th května 65
  postal-code = 14021
  locality = Прага 4
  country-name = Чехия
Item
  Id: __3
  Type: http://data-vocabulary.org/Geo
  latitude = 50.047893
  longitude = 14.4491
```

Как видите, здесь вся разметка налицо. Свойствам, выступающим как самостоятельные единицы микроданных, присвоены внутренние идентификаторы (`Item(__1)`, `Item(__2)` и т. д.), что не входит в состав спецификации микроданных. Тестовые инструменты Google пользуются этой условной формой вывода для отображения нелинейных структур: так более наглядно видна группировка вложенных единиц и их свойств.

Как же в выдаче Google может быть отображена наша страница-образец? Не забывайте, что это только пример: Google может в любой момент изменить формат выдачи результатов поиска, а может вообще не обратить внимания на вашу разметку микроданных. Приблизительный вид выдачи показан на рис. 10.2.

Mark Pilgrim's event calendar
Excerpt from the page will show up here.
Excerpt from the page will show up here.

Google Developer Day 2009	Fri, Nov 6	Congress Center, Praha 4, Czech Republic
ConFoo.ca 2010	Wed, Mar 10	Hilton Montreal Bonaventure, Montréal, Québec, Canada

diveintohtml5.org/examples/event-plus-microdata.html - Cached - Similar pages

Рис. 10.2. Образец поисковой выдачи, содержащей страницу, на которой информация о событии размечена микроданными

После названия страницы и выдержки из ее текста, которая выбирается автоматически, Google на основе нашей разметки микроданных отображает небольшую таблицу с событиями. Дата имеет вид: Fri, Nov 6. Где-либо в нашей HTML-разметке и в микроданных ничего подобного нет. Мы использовали две полных строки в формате ISO: 2009-11-06T08:30+01:00 и 2009-11-06T20:30+01:00. Google, сравнив две наши даты, заметил, что это один и тот же день, и вывел результат в виде единой даты, более удобной для пользовательского восприятия.

Теперь посмотрим на адреса событий. Google отображает адрес в формате: название учреждения + населенный пункт + страна (номер дома и название улицы игнорируются). Это возможно постольку, поскольку мы разбили адрес на пять самостоятельных свойств микроданных: `name`, `street-address`, `region`, `locality` и `country-name`. Google выводит не все пять, а только часть — сокращенный адрес. Другие системы, принимающие на вход тот же HTML-код с разметкой микроданных, могут выбрать для отображения другие свойства или иначе показывать их на экране; о «правильном» и «ошибочном» выборе здесь речи не идет. Веб-мастер должен предоставить как можно больше данных максимально возможной точности, а кто и как этими данными будет оперировать — целиком на их совести.

Разметка клиентских отзывов

Поговорим еще об одном способе улучшить веб-страницы (и, возможно, поисковую выдачу) с помощью микроданных. Речь пойдет об отзывах о товарах и услугах.

Вот краткий отзыв о моей любимой пиццерии, расположенной в том же районе, где я живу. Этот ресторан существует на самом деле. Если вам случится бывать в Апексе, штат Северная Каролина, — заходите, не пожалеете. Первоначально отзыв был сверстан так (http://diveintohtml5.org/examples/review.html):

```
<article>
  <h1>Anna's Pizzeria</h1>
  <p>★★★★☆ (4 звезды из 5 возможных)</p>
  <p>В историческом центре Апекса - пиццерия с нью-йоркской атмосферой</p>
  <p>
    Еда первоклассная. По духу заведение вполне отвечает формату "ближайшей
    к дому пиццерии". Ресторанный зал несколько тесноват, отчего полным
    людям трудно пробираться между столами к своим местам. Раньше бесплатно
    давали зубчики чеснока, а теперь только хлеб: за вкусное надо платить.
    В целом - отличное место.
  </p>
  <p>
    100 North Salem Street<br>
    Apex, NC 27502<br>
    USA
  </p>
  <p>— отзыв оставил Марк Пилгрим, последнее обновление 31 марта 2010 г.</p>
</article>
```



ПРИМЕЧАНИЕ

За изменениями, которые в этом разделе мы вносим в код страницы, вы можете проследить в режиме онлайн. Было: http://diveintohtml5.org/examples/review.html; стало: http://diveintohtml5.org/examples/review-plus-microdata.html.

Поскольку отзыв обернут тегом `<article>`, именно в этом теге мы объявим атрибуты `itemtype` и `itemscope`. Задающий пространство имен URL-адрес словаря выглядит так:

```
<article itemscope itemtype="http://data-vocabulary.org/Review">
```

Какие свойства доступны в словаре микроданных Review? Внимание на табл. 10.5.

Таблица 10.5. Словарь Review

Свойство	Описание
itemreviewed	Название услуги, товара, предприятия и т. п., которому посвящен отзыв
rating	Численная оценка качества по шкале от 1 до 5; для использования нестандартной шкалы нужен словарь Rating (http://data-vocabulary.org/Rating)
reviewer	Имя человека, который оставил отзыв
dtreviewed	Дата (в ISO-формате), когда был оставлен отзыв
summary	Краткое содержание отзыва
description	Основное содержание отзыва

Первое свойство легко сразу же применить: itemreviewed — обычный текст, который в нашем примере находится в заголовке <h1>. Здесь мы должны объявить атрибут itemprop с соответствующим значением:

```
<h1 itemprop="itemreviewed">Anna's Pizzeria</h1>
```

К вопросу о рейтинге я еще вернусь; пока оставим его без рассмотрения.

Следующие два свойства просты: summary кратко описывает товар или услугу с точки зрения лица, оставившего отзыв, а description — основной текст отзыва:

```
<p itemprop="summary">В историческом центре Апекса - пиццерия с нью-йоркской атмосферой</p>
<p itemprop="description">
  Еда первоклассная. По духу заведение вполне отвечает формату "ближайшей
  к дому пиццерии". Ресторанный зал несколько тесноват, отчего полным
  людям трудно пробираться между столами к своим местам. Раньше бесплатно
  давали зубчики чеснока, а теперь только хлеб: за вкусное надо платить.
  В целом - отличное место.</p>
```

Со свойствами location и geo нам, в сущности, уже приходилось сталкиваться (чтобы удостовериться в этом, просмотрите предыдущие разделы, где показано, как размечать микроданными адрес человека, адрес организации и геолокационную информацию):

```
<p itemprop="location" itemscope
   itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">100 North Salem Street</span><br>
  <span itemprop="locality">Apex</span>,
  <span itemprop="region">NC</span>
  <span itemprop="postal-code">27502</span><br>
  <span itemprop="country-name">USA</span>
</p>
  <span itemprop="geo" itemscope
        itemtype="http://data-vocabulary.org/Geo">
    <meta itemprop="latitude" content="35.730796" />
    <meta itemprop="longitude" content="-78.851426" />
  </span>
```

В последней строке наблюдается уже знакомая проблема: два самостоятельных фрагмента информации в одном элементе. Человека, который оставил отзыв, зовут

Марк Пилгрим, а дата, когда был оставлен отзыв, — 31 марта 2010 года. Как «разделить» два свойства микроданных? Как обычно, следует воспользоваться независимыми обертками (см. раздел «Разметка данных о человеке» этой главы), в каждой из которых объявить атрибут itemprop. В сущности, дату в этом образце следовало бы оформить тегом <time>, что и дало бы естественную обертку, на которую удобно «навесить» атрибут itemprop. Тогда достаточно вложить имя лица, оставившего отзыв, в тег <span>:

```
  <p>
    <span itemprop="reviewer">Марк Пилгрим</span>, последнее обновление -
    <time itemprop="dtreviewed" datetime="2010-03-31">
      31 марта 2010 г.
    </time>
  </p>
</article>
```

Теперь поговорим о рейтингах. В клиентском отзыве о товаре или услуге труднее всего правильно разметить рейтинг. По умолчанию словарь Review предполагает выставление рейтинга по шкале от 1 до 5, на которой 1 — это ужасно, а 5 — прекрасно. Вы можете, если хотите, пользоваться и другой шкалой, но прежде, чем перейти к таковой, рассмотрим стандартную рейтинговую шкалу:

```
<p>★★★★☆ (<span itemprop="rating">4</span> звезды из 5 возможных)</p>
```

Если вы применяете эту шкалу, то свойству микроданных itemprop="rating" надо присвоить числовое значение рейтинга (в данном случае 4) — и все. А если бы вы работали с другой шкалой? Тогда следует дополнительно объявить границы диапазона оценок. Так, при переходе к шкале 0... 10 разметка свойства itemprop="rating" останется прежней, но значение рейтинга будет теперь устанавливаться не напрямую, а через вложенный словарь микроданных Rating (http://data-vocabulary.org/Rating), который требует вдобавок объявлять худшее и лучшее значения на используемой шкале:

```
<p itemprop="rating" itemscope
   itemtype="http://data-vocabulary.org/Rating">
  ★★★★★★★★★☆
  (<span itemprop="value">9</span> по шкале от
   <span itemprop="worst">0</span> до
   <span itemprop="best">10</span>)
</p>
```

По-русски это значит: «Оцениваемому продукту я склонен присвоить рейтинг 9 по 10-балльной шкале».

Говорилось ли уже, что разметка микроданных над клиентскими отзывами позволяет влиять на поисковую выдачу? Если нет, то разговором об этом мы и завершим главу.

Вот «сырые» данные, которые были извлечены инструментами Google Rich Snippets из моего отзыва, оснащенного разметкой микроданных (http://www.google.com/webmasters/tools/richsnippets?url=//diveintohtml5.org/examples/review-plus-microdata.html):

```
Item
  Type: http://data-vocabulary.org/Review
  itemreviewed = Anna's Pizzeria
  rating = 4
  summary = В историческом центре Апекса - пиццерия с нью-йоркской атмосферой
  description = Еда первоклассна. По духу заведение вполне отвечает…
  address = Item(__1)
  geo = Item(__2)
  reviewer = Марк Пилгрим
  dtreviewed = 2010-03-31
Item
  Id: __1
  Type: http://data-vocabulary.org/Organization
  street-address = 100 North Salem Street
  locality = Apex
  region = NC
  postal-code = 27502
  country-name = USA
Item
  Id: __2
  Type: http://data-vocabulary.org/Geo
  latitude = 35.730796
  longitude = -78.851426
```

Образец поисковой выдачи, содержащей мой отзыв, будет выглядеть (если делать поправку на странности системы Google, фазы Луны и т. п.) так, как показано на рис. 10.3.

Anna's Pizzeria: review
☆☆☆☆☆ Review by Mark Pilgrim - Mar 31, 2010
Excerpt from the page will show up here.
Excerpt from the page will show up here.
diveintohtml5.org/examples/review-plus-microdata.html - Cached - Similar pages

Рис. 10.3. Образец поисковой выдачи, содержащей страницу, на которой клиентский отзыв размечен микроданными

Для дальнейшего изучения

О микроданных:

- Live microdata playground (http://foolip.org/microdatajs/live/);
- спецификация микроданных HTML5 (http://bit.ly/ckt9Rj).

О системе Google Rich Snippets:

- «О структурированных данных и обогащении поисковой выдачи» (http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170);
- «Люди» (http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146646);

- «Компании и организации» (http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146861);
- «События» (http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=164506);
- «Отзывы» (http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146645);
- «Рейтинги в отзывах» (http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=172705);
- инструментарий для тестирования Google Rich Snippets (http://www.google.com/webmasters/tools/richsnippets);
- «Советы и хитрости Google Rich Snippets» (http://knol.google.com/k/google-rich-snippets-tips-and-tricks).

Приложение. Универсальный почти алфавитный определитель всего на свете

Не понимаете, как этим пользоваться? Обратитесь к главе 2, где изложены начальные сведения о тестировании HTML5-функциональности. Определять все те же функции позволяет библиотека Modernizr (http://www.modernizr.com).

Элементы

- <audio> (http://bit.ly/cZxI7K):

```
return !!document.createElement(‘audio’).canPlayType;
```

- <audio> в формате MP3 (http://ru.wikipedia.org/wiki/MP3):

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/mpeg;').replace(/no/, ''));
```

- <audio> в формате Vorbis (http://ru.wikipedia.org/wiki/Vorbis):

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/ogg;
          codecs="vorbis"').replace(/no/, ''));
```

- <audio> в формате WAV (http://en.wikipedia.org/wiki/WAV):

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/wav;
          codecs="1"').replace(/no/, ''));
```

- <audio> в формате AAC (http://ru.wikipedia.org/wiki/Advanced_Audio_Coding):

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/mp4;
          codecs="mp4a.40.2"').replace(/no/, ''));
```

- <canvas> (см. главу 4):
```
return !!document.createElement('canvas').getContext;
```
- <canvas>, API рисования текста (см. раздел «Текст» главы 4):
```
var c = document.createElement('canvas');
return c.getContext && typeof c.getContext('2d').fillText == 'function';
```
- <command> (http://bit.ly/aQt2Fn):
```
return ‘type’ in document.createElement(‘command’);
```
- <datalist> (http://bit.ly/9WVz5p):
```
return ‘options’ in document.createElement(‘datalist’);
```
- <details> (http://bit.ly/cO8mQy):
```
return ‘open’ in document.createElement(‘details’);
```
- <device> (http://bit.ly/aaBeUy):
```
return ‘type’ in document.createElement(‘device’);
```
- <form>, валидация (http://bit.ly/cb9Wmj):
```
return 'noValidate' in document.createElement('form');
```
- <iframe sandbox> (http://blog.whatwg.org/whats-next-in-html-episode-2-sandbox):
```
return ‘sandbox’ in document.createElement(‘iframe’);
```
- <iframe srcdoc> (http://blog.whatwg.org/whats-next-in-html-episode-2-sandbox):
```
return ‘srcdoc’ in document.createElement(‘iframe’);
```
- <input autofocus> (см. «Автофокусировка полей» на с. 148):
```
return 'autofocus' in document.createElement('input');
```
- <input placeholder> (см. «Подсказывающий текст» на с. 147):
```
return 'placeholder' in document.createElement('input');
```
- <input type="color"> (http://bit.ly/9HkeNn):
```
var i = document.createElement(‘input’);
i.setAttribute(‘type’, ‘color’);
return i.type !== ‘text’;
```
- <input type="email"> (см. раздел «Адреса электронной почты» главы 9):
```
var i = document.createElement('input');
i.setAttribute('type', 'email');
return i.type !== 'text';
```
- <input type="number"> (см. раздел «Числа как счетчики» главы 9):
```
var i = document.createElement('input');
```

```
i.setAttribute('type', 'number');
return i.type !== 'text';
```

- `<input type="range">` (см. раздел «Числа как ползунки» главы 9):

```
var i = document.createElement('input');
i.setAttribute('type', 'range');
return i.type !== 'text';
```

- `<input type="search">` (см. раздел «Формы поиска» главы 9):

```
var i = document.createElement('input');
i.setAttribute('type', 'search');
return i.type !== 'text';
```

- `<input type="tel">` (http://bit.ly/bZm0Q5):

```
var i = document.createElement(‘input’);
i.setAttribute(‘type’, ‘tel’);
return i.type !== ‘text’;
```

- `<input type="url">` (см. раздел «Веб-адреса» главы 9):

```
var i = document.createElement('input');
i.setAttribute('type', 'url');
return i.type !== 'text';
```

- `<input type="date">` (см. раздел «Выборщики даты» главы 9):

```
var i = document.createElement('input');
i.setAttribute('type', 'date');
return i.type !== 'text';
```

- `<input type="time">` (см. раздел «Выборщики даты» главы 9):

```
var i = document.createElement('input');
i.setAttribute('type', 'time');
return i.type !== 'text';
```

- `<input type="datetime">` (см. раздел «Выборщики даты» главы 9):

```
var i = document.createElement('input');
i.setAttribute('type', 'datetime');
return i.type !== 'text';
```

- `<input type="datetime-local">` (см. раздел «Выборщики даты» главы 9):

```
var i = document.createElement('input');
i.setAttribute('type', 'datetime-local);
```

```
return i.type !== 'text';
```

- <input type="month"> (см. раздел «Выборщики даты» главы 9):

```
var i = document.createElement('input');
i.setAttribute('type', 'month');
return i.type !== 'text';
```

- <input type="week"> (см. раздел «Выборщики даты» главы 9):

```
var i = document.createElement('input');
i.setAttribute('type', 'week');
return i.type !== 'text';
```

- <meter> (http://bit.ly/c0pX0l):

```
return ‘value’ in document.createElement(‘meter’);
```

- <output> (http://bit.ly/asJaqH):

```
return ‘value’ in document.createElement(‘output’);
```

- <progress> (http://bit.ly/bjDMy6):

```
return ‘value’ in document.createElement(‘progress’);
```

- <time> (http://bit.ly/bI62jp):

```
return ‘valueAsDate’ in document.createElement(‘time’);
```

- <video> (см. главу 5):

```
return !!document.createElement('video').canPlayType;
```

- <video>, заголовки (http://bit.ly/9mLiRr):

```
return 'track' in document.createElement('track');
```

- <video poster> (http://bit.ly/b6RhzT):

```
return ‘poster’ in document.createElement(‘video’);
```

- <video> в формате WebM (http://www.webmproject.org):

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/webm; codecs="vp8,
          vorbis"').replace(/no/, ''));
```

- <video> в формате H.264 (см. раздел «Видеокодеки» главы 5):

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/mp4; codecs="avc1.42E01E,
          mp4a.40.2"').replace(/no/, ''));
```

- <video> в формате Theora (см. раздел «Видеокодеки» главы 5):

```
var v = document.createElement('video');
```

```
return !!(v.canPlayType && v.canPlayType('video/ogg; codecs="theora,
          vorbis"').replace(/no/, ''));
```

- `contentEditable` (http://bit.ly/aLivbS):

```
return 'isContentEditable' in document.createElement('span');
```

- Drag-and-drop (http://bit.ly/aN0RFQ):

```
return 'draggable' in document.createElement('span');
```

- SVG (http://www.w3.org/TR/SVG/):

```
return !!(document.createElementNS && document.createElementNS
('http://www.w3.org/2000/svg', 'svg').createSVGRect);
```

- SVG в `text/html` (http://hacks.mozilla.org/2010/05/firefox-4-the-html5-parser-inline-svg-speed-and-more/):

```
var e = document.createElement('div');
e.innerHTML = '<svg></svg>';
return !!(window.SVGSVGElement && e.firstChild instanceof
          window.SVGSVGElement);
```

- WebSimpleDB (http://dev.w3.org/2006/webapi/WebSimpleDB/):

```
return !!window.indexedDB;
```

- Web Sockets (http://dev.w3.org/html5/websockets/):

```
return !!window.WebSocket;
```

- Web SQL (http://dev.w3.org/html5/webdatabase/):

```
return !!window.openDatabase;
```

- Геолокация (см. главу 6):

```
return !!navigator.geolocation;
```

- История (http://bit.ly/9JGAGB):

```
return !!(window.history && window.history.pushState &&
window.history.popState);
```

- Локальное хранилище (http://dev.w3.org/html5/webstorage/):

```
return ('localStorage' in window) && window['localStorage'] !== null;
```

- Микроданные (http://bit.ly/dBGnqr):

```
return !!document.getItems;
```

- Обмен сообщениями между документами (cross-document messaging) (http://bit.ly/cUOqXd):

```
return !!window.postMessage;
```

- Офлайновые веб-приложения (см. главу 8):

```
return !!window.applicationCache;
```

- Сессионное хранилище (http://dev.w3.org/html5/webstorage/):

```
try {
  return ('sessionStorage' in window) && window['sessionStorage'] !== null;
} catch(e) {
  return false;
}
```

- События, посылаемые сервером (server-sent events) (http://dev.w3.org/html5/eventsource/):

```
return typeof EventSource !== 'undefined';
```

- Файловый API (http://dev.w3.org/2006/webapi/FileAPI/):

```
return typeof FileReader != 'undefined';
```

- Фоновые вычисления (Web Workers) (http://bit.ly/9jheof):

```
return !!window.Worker;
```

- Undo (http://bit.ly/bs6JFR):

```
return typeof UndoManager !== ‘undefined’;
```

Для дальнейшего изучения

Спецификации и стандарты:

- HTML5 (http://bit.ly/bYiOQp);
- геолокация (http://www.w3.org/TR/geolocation-API/);
- Server-Sent Events (http://dev.w3.org/html5/eventsource/);
- WebSimpleDB (http://dev.w3.org/2006/webapi/WebSimpleDB/);
- Web Sockets (http://dev.w3.org/html5/websockets/);
- Web SQL Database (http://dev.w3.org/html5/webdatabase/);
- локальное хранилище (http://dev.w3.org/html5/webstorage/);
- фоновые вычисления (http://bit.ly/9jheof).

JavaScript-библиотека Modernizr (http://www.modernizr.com) — библиотека для тестирования HTML5-функций.